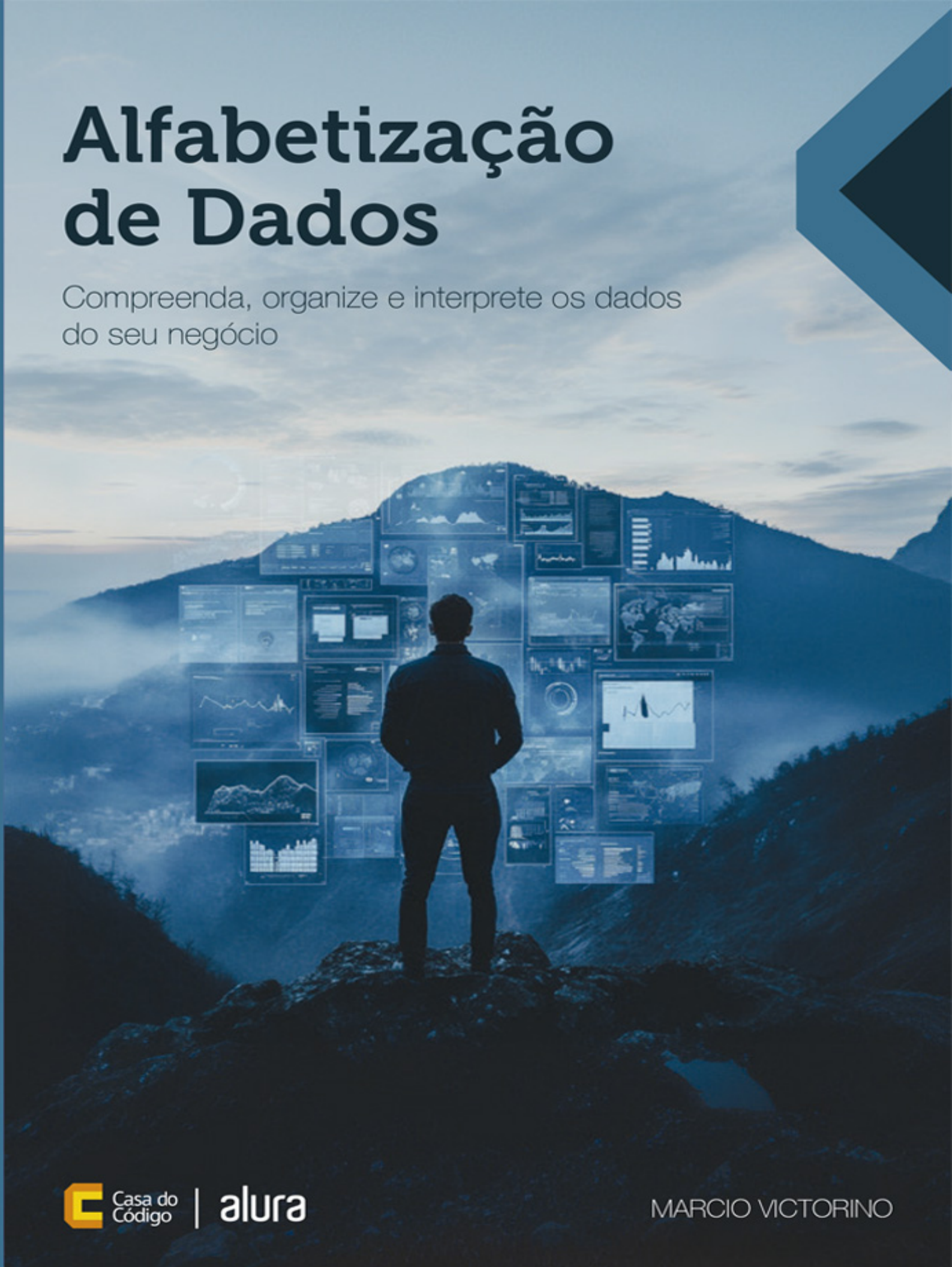
Alfabetização de Dados

Compreenda, organize e interprete os dados do seu negócio



Casa do Código | alura

MARCIO VICTORINO

Sumário

- ISBN
- Sobre o autor
- Dedicatória
- Prefácio
- 1. Alfabetização de dados
- 2. Sistemas de informação
- 3. Projeto conceitual de banco de dados
- 4. Projeto lógico de banco de dados
- 5. Projeto físico de banco de dados
- 6. Big Data e banco de dados NoSQL
- 7. Data Warehouse
- 8. Lógica de Programação
- 9. Linguagem de programação Python
- 10. Ciência de Dados
- 11. Aprendizado supervisionado
- 12. Aprendizado não supervisionado
- 13. Considerações finais

ISBN

Impresso: 978-85-5519-386-6

Digital: 978-85-5519-387-3

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Victorino, Marcio
Alfabetização de dados : compreenda, organize e interprete os dados do seu negócio / Marcio Victorino. -- 1. ed. -- São Paulo : AOVS Sistemas de Informática, 2024.

ISBN 978-85-5519-386-6

1. Algoritmos de computadores 2. Ciência da computação 3. Dados - Análise 4. Dados - Estruturas (Ciência da computação) 5. Linguagem de programação (Computadores) 6. Processamento de dados I. Título.

24-232186 CDD-005.73

Índices para catálogo sistemático:

1. Dados : Estruturas : Processamento de dados 005.73

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

Sobre o autor

Marcio Victorino é engenheiro da computação formado pelo Instituto Militar de Engenharia (IME), possui especialização em Sistemas de Informação e Telemática pela Universidade Federal do Rio Grande do Sul (UFRGS), mestrado em Sistemas e Computação pelo IME e doutorado em Ciência da Informação pela Universidade de Brasília (UnB).

Tornou-se profissional de Tecnologia da Informação (TI) em 1994 quando concluiu o curso de graduação. Desde então, participou de vários projetos de TI do Exército Brasileiro (EB), desempenhando diversos papéis, iniciando como desenvolvedor de software até assumir a posição de gerente de projetos.

Em paralelo às atividades de engenharia, dedicou-se à carreira acadêmica por acreditar que a experiência adquirida no desenvolvimento de projetos de TI seria de grande valia para a formação de futuros profissionais da área. Em 2002, começou a ministrar disciplinas de TI em instituições de ensino superior do Distrito Federal. Em 2014, quando passou para a reserva do EB, ingressou como professor substituto do Departamento de Ciência da Computação (CIC) e, em 2017, tornou-se professor adjunto da Faculdade de Ciência da Informação (FCI) da UnB. Atualmente, é professor da Graduação em Biblioteconomia da FCI e participa de dois programas de Pós-Graduação da UnB nas seguintes linhas de pesquisa: Gestão, Tecnologias e Organização da Informação e do Conhecimento do PPGCInf (FCI) e Ciência de Dados do PPCA (CIC).

Mais informações podem ser encontradas em seu perfil no LinkedIn: https://www.linkedin.com/in/marcio-victorino-unb/ e no seu currículo Lattes: http://lattes.cnpq.br/7851064698387435.

Dedicatória

Dedico este livro à Lilian, mulher que eu nunca pedi a Deus, pois jamais imaginaria uma pessoa tão completa. Quando estava no momento mais conturbado da minha vida, eu só buscava por paz, aí Deus me apresentou a ela, mulher forte, decidida, inteligente, companheira, linda e bem-humorada. Quando descobri que seu nome completo é Lilian B. Paz, casei! Essa era a Paz que eu nem sabia que procurava e ao encontrá-la obtive a tranquilidade e o incentivo fundamentais para escrever este livro.

Agradecimentos

A Deus, responsável por tudo.

À Dona Lia e ao Seu Gilberto (*in memoriam*) por serem os melhores pais que Deus poderia ter me concedido nesta encarnação.

À Lilian, minha esposa, por tentar me convencer diariamente que sou um homem melhor do que eu penso ser. Espero um dia me tornar o homem que ela pensa que sou.

Ao casal Cláudia e Cláudio Paranhos (*in memoriam*), pelo suporte dado durante o curso de graduação do IME. Serei eternamente grato ao Paranhos por ter me arrastado para estudar durante a nossa graduação nos dias em que eu não estava com disposição. Muito obrigado por tudo, meu grande amigo!

Prefácio

"*Viver é enfrentar um problema atrás do outro. O modo como você os encara é que faz a diferença*". Essa afirmação é atribuída a Benjamin Franklin, um dos mais importantes personagens da história dos Estados Unidos e mundialmente conhecido por suas pesquisas científicas na área da eletricidade.

Alinhados ao pensamento de Franklin, mapeando-o para o ambiente corporativo, podemos afirmar que gerir uma organização é enfrentar um problema atrás do outro. O modo como a pessoa gestora o encara é que faz a diferença. A eficácia de uma pessoa gestora é fundamentada pela sua capacidade de tomar decisões para resolver os problemas de uma organização.

Em um mundo no qual as organizações são ou estão tentando se tornar orientadas a dados (*data driven*), a diferença entre o bom gestor e o não tão bom gestor é que o primeiro tende a resolver os problemas da organização tomando decisões com base nos dados, enquanto o outro tende a não dar o valor devido aos dados para a resolução desses problemas. Esses comportamentos distintos talvez possam ser explicados pela capacidade que cada gestor tem de compreender, organizar e interpretar os dados de uma corporação.

As organizações orientadas a dados atuam em um ambiente repleto de sistemas de informação transacionais e analíticos, que geram, coletam e mantêm dados sobre suas operações, clientes, concorrentes e mercado. Esses sistemas e respectivos dados gerados são subsídios fundamentais para a tomada de decisão.

A boa notícia é que esse ambiente organizacional complexo gera dados relevantes a todo momento, potencialmente utilizáveis para o entendimento do comportamento dos clientes, do negócio da organização e da concorrência. Por outro lado, devido ao grande volume, à grande variedade de formatos e à velocidade com que

esses dados devem ser processados para não se tornarem obsoletos, há a necessidade de se ter um conhecimento sobre a linguagem dos dados, independentemente da área de atuação e posição ocupada nas organizações. Essa linguagem se refere à facilidade para ler e interpretar visualizações de dados e entender os dados subjacentes.

Pode-se inferir que a qualidade dos sistemas de informação e a capacidade de pessoas gestoras fazerem o uso apropriado dos dados gerados por esses sistemas podem impactar dramaticamente a qualidade da gestão dessas organizações.

Então, surge a indagação: *Como aprimorar a qualidade dos sistemas de informação e a compreensão dos dados gerados por esses sistemas?*

Durante os últimos 30 anos, tive a oportunidade de participar de vários projetos que incluíam a concepção, implementação e implantação de sistemas de informação transacionais e analíticos. Nesse período, foi possível perceber que, quanto mais eu explicava aos usuários como funcionava o processo de desenvolvimento de sistemas de informação computadorizados e as tecnologias utilizadas, maior era o interesse deles em relação a esses assuntos. Esse círculo virtuoso incrementava significativamente a qualidade dos sistemas de informação e a compreensão dos dados gerados por esses sistemas.

Tudo começou em 1994, quando concluí o curso de Engenharia da Computação e comecei a fazer parte de equipes de projetos do Exército Brasileiro (EB) para o desenvolvimento de sistemas computacionais. Mais tarde, em 2002, após concluir o mestrado, iniciei a carreira acadêmica em paralelo à carreira de profissional de Tecnologia da Informação (TI), ministrando aulas em instituições de ensino superior do Distrito Federal, mais precisamente em Brasília. Foram vários anos lidando com alunos e usuários dos mais diversos tipos de sistemas.

Sempre incentivei os usuários a entenderem o processo de desenvolvimento de sistemas. Com o passar do tempo, alguns usuários começaram a solicitar cursos de modelagem de dados para que eles entendessem melhor os modelos conceituais dos sistemas a serem implementados. Atendi a essas demandas e foi dessa forma que iniciei a ensinar TI para profissionais que não são da área de tecnologia. Assim, foi possível perceber que quanto mais os usuários entendiam o processo de desenvolvimento de software, mais eles participavam do projeto de desenvolvimento e, consequentemente, os sistemas desenvolvidos apresentavam maior qualidade e esses usuários compreendiam profundamente os dados gerados por esses sistemas.

Particularmente, sempre acreditei que os profissionais de qualquer área de formação ou atuação são capazes de entender os fundamentos dos sistemas de informação computadorizados e as tecnologias envolvidas, até que, no início da década de 2010, tive contato com o termo **alfabetização de dados** (*Data Literacy*), consolidando a certeza de que é possível e útil ensinar os fundamentos dos processos e tecnologias utilizadas para o tratamento dos dados para esses profissionais.

> **Alfabetização de dados** pode ser entendida como a capacidade de entender, encontrar, ler, interpretar, avaliar, gerenciar e usar dados. Ela permite que os indivíduos acessem, interpretem, avaliem criticamente, gerenciem, manejem e usem os dados eticamente (PRADO; MARZAL, 2013)*.

Então, para uma pessoa profissional ser considerada *alfabetizada em dados*, ela precisa desenvolver um conjunto de habilidades relacionadas a processos e tecnologias computacionais utilizadas para o tratamento da informação. O principal objetivo deste livro é proporcionar o entendimento da linguagem dos dados, de como esses dados devem ser organizados e como podem ser

transformados para serem usados no apoio à decisão nas organizações.

Para incentivar o processo de alfabetização de dados de pessoas universitárias e profissionais de qualquer área, resolvi escrever este livro, que é um registro da experiência adquirida nos últimos 30 anos por meio da participação em projetos de TI e atuação como professor de disciplinas em cursos de graduação e pós-graduação na área de Ciência da Computação e Ciência da Informação, ministradas para universitários e profissionais de diversas áreas de formação e atuação.

Para enriquecer sua experiência durante a leitura, você pode visualizar a implementação dos principais conceitos apresentados neste livro também por meio do canal do YouTube ***@ProgramarIsCool*** e obter materiais complementares no repositório ***LivroAlfabetizacaoDeDados*** (https://github.com/profvictorino/LivroAlfabetizacaoDeDados) do usuário ***profvictorino*** (https://github.com/profvictorino) do **GitHub**.

Para desenvolver as principais habilidades preconizadas pela alfabetização de dados, este livro está organizado da seguinte forma:

- Capítulo 1: Alfabetização de dados
- Capítulo 2: Sistemas de informação
- Capítulo 3: Projeto conceitual de banco de dados
- Capítulo 4: Projeto lógico de banco de dados
- Capítulo 5: Projeto físico de banco de dados
- Capítulo 6: Big Data e banco de dados NoSQL
- Capítulo 7: Data Warehouse
- Capítulo 8: Lógica de Programação
- Capítulo 9: Linguagem de programação Python
- Capítulo 10: Ciência de Dados
- Capítulo 11: Aprendizado supervisionado
- Capítulo 12: Aprendizado não supervisionado

- Capítulo 13: Considerações finais

---

* PRADO, Javier Calzada; MARZAL, Miguel Ángel. Incorporating data literacy into information literacy programs: Core competencies and contents. **Libri**, v. 63, n. 2, p. 123-134, 2013.

CAPÍTULO 1
Alfabetização de dados

"Tudo o que temos a decidir é o que fazer com o tempo que nos é concedido". Essa afirmação do personagem Gandalf do filme *O Senhor dos Anéis* pode ser perfeitamente adaptada para a gestão corporativa no contexto em que vivemos para *"Tudo o que temos a decidir é o que fazer com os dados que nos são dados"*.

Indubitavelmente, os dados são um ativo extremamente valioso para as organizações. Clive Humby, matemático londrino especializado em Ciência de Dados, afirma que "*data is the new oil*" ("dados são o novo petróleo"). Essa expressão tem sido citada com grande frequência no ambiente corporativo. Gestores dos mais diversos tipos de organizações no Brasil e no exterior usam-na para difundir a ideia de que os dados são tão valiosos quanto o petróleo, ou seja, as pessoas gestoras que souberem obter valor (*insights*) a partir dos dados para aprimorar suas decisões serão detentoras de um grande diferencial competitivo.

Na verdade, a afirmação original de Humby é mais completa: "*Data is the new oil. It's valuable, but if unrefined it cannot really be used (...) so must data be broken down, analyzed for it to have value*". Em tradução livre, "*Os dados são o novo petróleo. São valiosos, mas, se não forem refinados, não podem realmente ser usados (...), então os dados devem ser pormenorizados e analisados para que tenham valor.*".

Nessa afirmação, o autor defende que, tal como o petróleo, o dado precisa ser refinado e analisado para que se obtenha algo valioso. Assim, profissionais e organizações precisam, cada vez mais, ser alfabetizados em dados para falarem, de fato, a linguagem de dados e conseguirem usá-los em seu potencial máximo.

A maior riqueza não está nos dados em si, mas na capacidade de usá-los de forma analítica. No entanto, diferentemente do petróleo, localizar dados não é um problema, tendo em vista que eles estão à disposição em todo o ambiente corporativo e fora dele. O maior desafio é saber como fazer bom uso dessa fonte abundante de possibilidades. Enquanto o petróleo é escasso, finito e de uso específico, os dados não são, e suas possibilidades de uso são inesgotáveis.

Por essa razão, as organizações convencionais estão se esforçando para se transformarem em organizações orientadas a dados. As organizações orientadas a dados atuam em um ambiente repleto de sistemas de informação transacionais e analíticos, que geram, coletam e mantêm dados sobre suas operações, clientes, concorrentes e mercado para subsidiar suas decisões.

Independente de qual tecnologia da informação seja usada para a automação dos sistemas de informação, todas elas têm os dados como matéria-prima. A execução dos processos dessas organizações é feita com base nos dados coletados e gerados. Assim, os dados são a base para a tomada de decisões, em vez de as organizações serem geridas intuitivamente.

Essa mudança estratégica exige que os integrantes das organizações, independente de área de atuação, saibam lidar com os dados. Nesse ambiente, as habilidades de alfabetização de dados são importantes para os colaboradores em todas as áreas de uma organização.

Existem várias definições para o termo **alfabetização de dados**. Dentre elas, podemos destacar a de Wolff *et al*. (2016), que explicita muito bem as habilidades desejáveis a qualquer profissional em organizações orientadas a dados. Os autores definem alfabetização de dados como a capacidade de fazer e responder perguntas do mundo real a partir de grandes ou pequenos conjuntos de dados por meio de um processo de consulta, respeitando o uso ético dos dados. Baseia-se nas principais habilidades práticas e criativas, com

a capacidade de ampliar o conhecimento das habilidades especializadas em manipulação de dados de acordo com os objetivos. Isso inclui a capacidade de selecionar, limpar, analisar, visualizar, criticar e interpretar dados, além de comunicar histórias a partir de dados e usá-los como parte da tomada de decisão.

As pesquisadoras Burress, Mann e Neville (2020) desenvolveram um estudo, com a colaboração de bibliotecárias da University of South Florida St. Petersburg, que foca na importância da alfabetização de dados para o aperfeiçoamento da formação profissional de estudantes universitários de diversas áreas.

Segundo esse estudo, o impacto do advento do *Big Data* nas disciplinas acadêmicas criou a necessidade de todos os alunos terem alguma facilidade para ler e interpretar visualizações de dados e entender os dados subjacentes. Como resultado do estudo, a alfabetização de dados uniu-se a outras disciplinas para se tornar uma competência essencial da educação de graduação.

Em 2019, o Gartner, líder mundial em pesquisa e aconselhamento para empresas, divulgou recomendações para líderes de TI impulsionarem a alfabetização de dados dentro das suas organizações. Valerie Logan, vice-presidente de pesquisa do Gartner, afirmou que (WODONIS, 2019):

"A prevalência dos dados e dos recursos de análise de informações, incluindo Inteligência Artificial (IA), exige que criadores e consumidores 'falem a língua dos dados' como uma linguagem comum. Líderes de Data & Analytics devem defender a Alfabetização de Dados da força de trabalho como um facilitador de negócios digitais e tratar informações como uma segunda língua".

Os autores Mandinach e Gummer (2013) citam como habilidades almejadas pela alfabetização de dados saber como identificar, coletar, organizar, analisar, resumir e priorizar dados. A capacidade de desenvolver hipóteses, identificar problemas e interpretar os dados também são habilidades desejáveis.

Com base nas afirmações anteriores e na minha experiência de quase 30 anos atuando como profissional e professor universitário da área de TI, concordo que profissionais e universitários de qualquer área de formação devem passar pelo processo de alfabetização de dados a fim de desenvolverem um conjunto mínimo de habilidades para serem capazes de tomar decisões baseadas em dados.

A seguir, são elencadas as habilidades mais importantes à alfabetização de dados que serão abordadas neste livro:

- **Entender os sistemas de informação computadorizados:** o ponto de partida para entender os dados em uma organização é entender como esses dados são gerados, organizados e armazenados. Para isso, torna-se de fundamental importância entender as categorias dos sistemas de informação, seus objetivos e como eles são estruturados.
- **Entender projeto de bancos de dados relacionais:** os bancos de dados relacionais são muito utilizados como repositórios de dados das organizações, entender como esses dados são organizados no momento da sua persistência (gravação) é essencial para criarmos estratégias para a extração, limpeza e combinação dos dados oriundos de diversas fontes para a exibição nos mais variados formatos. Também, é muito importante entender as etapas de um projeto de banco de dados.
- **Entender os big data e bancos de dados NoSQL:** o principal objetivo do *big data* é proporcionar um ambiente apropriado à análise de grandes volumes de dados distribuídos para a geração de informações valiosas e, consequentemente, *insights* que auxiliarão as organizações no momento da tomada de decisão. Por outro lado, os bancos de dados NoSQL foram implementados para armazenar, recuperar e processar, de forma

eficaz, a grande quantidade de dados estruturados e não estruturados disponíveis no ambiente de *big data*. Portanto, é crucial compreender essas tecnologias e seu uso combinado em organizações orientadas a dados.

- **Entender data warehouse:** as organizações precisam utilizar informações de várias fontes de dados internas e externas para dar suporte ao negócio, entretanto, na maioria dos casos, os dados de interesse da organização não estão integrados e organizados no formato apropriado para o apoio à decisão. Para se tornarem relevantes eles precisam ser tratados. Uma estratégia muito utilizada para coletar, transformar, armazenar e apresentar os dados para o apoio à decisão é a criação de um *Data Warehouse* (DW).

- **Entender os fundamentos de lógica de programação:** a lógica de programação é o fundamento de qualquer linguagem de programação. Embora o nosso objetivo não seja formar programadores ou desenvolvedores de sistema, esse aprendizado será de grande valia para a criação de rotinas de tratamento de dados, como a extração, limpeza, transformação e combinação de dados oriundos de várias fontes.

- **Entender como os fundamentos de lógica de programação são implementados em uma linguagem de programação:** em computação, não basta lermos algo para aprendermos, é necessário pôr em prática. Aqui, usaremos a linguagem Python para colocarmos em prática o aprendizado de lógica de programação.

- **Entender os fundamentos de Ciência de Dados:** as organizações orientadas a dados, além de analisarem o seu passado e presente por meio dos sistemas de apoio a decisão, como por exemplo a tecnologia de DW, elas agora se esforçam, também, para prever (estimar) possíveis cenários futuros com uma determinada probabilidade de acerto por meio do uso das técnicas de Ciência de Dados. Então, serão apresentados os principais conceitos relacionados à Ciência de Dados e como são

implementadas as soluções mais comuns dessa área. Assim, o leitor ou leitora poderá entender modelos preditivos e descritivos que auxiliarão na tomada de decisão.

Repare que utilizei o termo "entender" para descrever as habilidades, ou seja, o(a) profissional alfabetizado(a) em dados não precisa ser um(a) especialista nas áreas elencadas anteriormente. Não há a intenção de que essa pessoa planeje ou desenvolva soluções de TI, porém ela deve entender os problemas, as técnicas e as tecnologias relacionadas ao tratamento dos dados corporativos. Acredito que esse(a) profissional será capaz de interagir com consultores técnicos de maneira consistente em busca de soluções que poderão gerar *insights* baseados nas informações corporativas, eliminando uma grande gama de problemas relacionados à comunicação entre profissionais do negócio e equipes técnicas de computação.

Por outro lado, também acredito que leitores e leitoras mais entusiasmados com as ferramentas de computação poderão, durante a leitura deste livro, desenvolver a capacidade de extrair, limpar, transformar, organizar dados e gerar relatórios analíticos para subsidiar as suas decisões, ou até mesmo implementar soluções que fazem uso de predição por meio de técnicas de Ciência de Dados.

Antes de encerrar este capítulo, gostaria de ressaltar a importância da alfabetização de dados por meio de uma analogia do ambiente corporativo com a indústria automobilística.

O brasileiro é reconhecidamente apaixonado por carro, e é possível observar que a maioria esmagadora dos proprietários de automóveis recorre a uma oficina mecânica quando o carro apresenta algum problema em vez de resolvê-lo na garagem de sua casa. No entanto, grande parte desses motoristas é capaz de explicar como funciona todos os subsistemas do seu automóvel: sabe diferenciar um carro antigo que trabalha com carburação de um mais moderno que possui injeção eletrônica; é capaz de diferenciar o câmbio manual do

automático; a tração dianteira da tração traseira; um carro 4x2 do 4x4, e assim por diante.

Na verdade, para dirigir um carro com câmbio automático, basta saber como ligar, engatar a marcha e acelerar. No entanto, quanto mais conhecermos sobre o funcionamento do carro e de seus subsistemas, melhor será o seu rendimento e a nossa percepção quanto ao surgimento de problemas que possam colocar em risco a normalidade de um deslocamento.

Assim, pode-se concluir que, quanto mais uma pessoa entender os sistemas integrantes e o funcionamento de um carro, mais segura será a sua direção, embora ela não possua a intenção de consertar ou até mesmo construir o seu próprio carro. Esse aprendizado também é muito útil quando o motorista explica a um mecânico os problemas do seu carro: eles conseguem utilizar uma linguagem comum, facilitando a comunicação entre os dois.

Por analogia, embora um(a) gestor(a) de uma organização orientada a dados nunca venha a desenvolver um sistema de informação, quanto mais ele(a) entender a respeito dos sistemas de informação e como a informação é tratada na organização, mais segura será a condução dessa organização. Esse aprendizado também facilitará a comunicação entre profissionais da área do negócio e profissionais de TI no planejamento da estratégia informacional da organização.

Agora que sabemos o que é alfabetização de dados, sua importância no ambiente corporativo e quais habilidades pretendemos desenvolver por meio deste livro, nos próximos capítulos veremos, de maneira didática, o que é preciso para desenvolver essas habilidades.

Referências

WODONIS, Daniela. Gartner apresenta recomendações para líderes de TI impulsionarem a

alfabetização de dados dentro de suas empresas. *Assespro-PR*, 2019. Disponível em: https://www.assespropr.org.br/gartner-apresenta-recomendacoes-para-lideres-de-ti-impulsionarem-a-alfabetizacao-de-dados-dentro-de-suas-empresas/. Acesso em: 15 mai. 2023.

BURRESS, Theresa; MANN, Emily; NEVILLE, Tina. Exploring data literacy via a librarian-faculty learning community: A case study. *The Journal of Academic Librarianship*, v. 46, n. 1, p. 102076, 2020. Disponível em: https://doi.org/10.1016/j.acalib.2019.102076. Acesso em: 6 set. 2023.

MANDINACH, Ellen B.; GUMMER, Edith S. A Systemic View of Implementing Data Literacy in Educator Preparation. *Educational Researcher*, v. 42, n. 1, p. 30-37, 2013. Disponível em: https://journals.sagepub.com/doi/epub/10.3102/0013189X12459803. Acesso em: 6 set. 2023.

WOLFF, Annika; GOOCH, Daniel; MONTANER, Jose J. C.; RASHID, Umar; KORTUEM, Gerd. Creating an Understanding of Data Literacy for a Data-driven Society. *The Journal of Community Informatics*, v. 12, n. 3, 2016. Disponível em:

https://openjournals.uwaterloo.ca/index.php/JoCI/article/view/3275/4298. Acesso em: 6 set. 2023.

Capítulo 2
Sistemas de informação

Já afirmava Steve Jobs: "*stay hungry, stay foolish*" (em português, *continue faminto, continue tolo*). A fome de um gestor pelo aprendizado deve ser eterna, a ponto de nunca se considerar um especialista tão completo para achar que não exista algo novo a aprender. Sendo assim, por mais que o gestor seja um especialista em sua área, convidamos você a mergulhar no mundo da Alfabetização de Dados, iniciando pelo entendimento dos conceitos relativos aos sistemas de informação.

Antes de apresentarmos uma definição para sistema de informação, vamos entender o que é *sistema* e o que é *informação*.

Com base na teoria geral de sistemas idealizada por Ludwig Von Bartalanffy, Jaime Robredo (2003) disponibiliza, em seu livro *Da Ciência da Informação Revisitada aos Sistemas Humanos de Informação*, uma variedade de definições para **sistema**:

- É um conjunto que funciona como um todo em virtude da interação de suas partes ou, mais simplesmente, um "pacote de relações";
- É qualquer coisa maior que a soma de suas partes, porque consta dessas partes mais da forma como elas se relacionam entre si, e mais também das qualidades que emergem dessas relações;
- É um conjunto de relações interativas, uma entidade relativamente bem identificada, que mantém em operação, dinamicamente, um certo todo;
- É o resultado inevitável de intenções organizadas, quer físicas, biológicas, psicológicas, sociológicas ou simbólicas;
- É um complexo de componentes que se torna uma entidade através das interações mútuas de suas partes, do átomo ao cosmos;

- É uma relação organizada das partes de um todo.

Por outro lado, podemos encontrar centenas ou até mesmo milhares de definições para **informação**, porém, para atingirmos os objetivos deste livro, escolhemos dois trabalhos para esclarecermos o termo no nosso contexto. O primeiro é de Urdaneta (1992), que sugere quatro classes diferentes de informação: *dados*, *informação*, *conhecimento* e *inteligência*:

- **Dados** são sinais que não foram processados, correlacionados, integrados, avaliados ou interpretados de qualquer forma, ou seja, é a matéria-prima a ser utilizada na produção da *informação*;

- A **informação** consiste nos dados processados para serem exibidos de forma inteligível às pessoas que vão utilizá-los;

- O **conhecimento** pode ser definido como sendo informações que foram analisadas e avaliadas quanto a sua confiabilidade, relevância e importância. Ele não é estático, modificando-se mediante interação com o ambiente;

- Já a **inteligência** pode ser entendida como sendo o conhecimento contextualmente relevante, que permite atuar com vantagens no ambiente considerado. Também pode ser vista como o conhecimento que foi sintetizado e aplicado a uma determinada situação, para ganhar maior profundidade de consciência.

O segundo trabalho, de McGee e Prusak (1994), está mais alinhado aos nossos objetivos. Os autores afirmam que a informação não se limita a dados coletados. Na verdade, *informação* são dados coletados, organizados, ordenados, aos quais são atribuídos significados e contexto. Ou seja, para que os dados se tornem úteis como informação a uma pessoa encarregada do processo decisório, é preciso que sejam apresentados de tal forma que essa pessoa possa relacioná-los e atuar sobre eles.

Após entendermos o que é sistema e o que é informação, podemos partir para a definição de sistema de informação proposta por Robredo (2003). Segundo o autor, **sistema de informação** pode ser entendido como uma entidade complexa, e organizada, que capta, armazena, processa, fornece, usa e distribui a informação.

A partir dessa definição, é possível concluir que uma organização possui diversos sistemas de informação, grande parte deles já automatizada por meio de ferramentas computacionais, dando origem aos sistemas de informação computadorizados, nosso principal foco. Esses sistemas podem usar a informação com propósitos diferentes, o que acaba originando categorias distintas de sistemas de informação. A seguir, apresentaremos uma classificação para os sistemas de informação.

2.1 Classificação dos sistemas de informação

Existem várias classificações para sistemas de informação, e diversos autores sugerem várias categorias diferentes. No entanto, abordaremos apenas as duas categorias de sistemas de informação computadorizados mais importantes: os *sistemas transacionais* e os *sistemas de apoio à decisão*.

Sistemas transacionais

Os sistemas transacionais encontram-se no nível operacional das organizações. Esses sistemas são os de mais baixo nível hierárquico em uma organização; eles atendem às necessidades operacionais e automatizam tarefas altamente estruturadas. Suas informações sofrem atualizações constantes e normalmente são armazenadas de maneira estruturada, por exemplo, em tabelas compostas por linhas e colunas.

Como exemplo de sistemas transacionais, podemos citar os sistemas de pagamento de salários ou sistemas de emissão de notas fiscais nas empresas, sistemas de matrícula de alunos em instituições de ensino, sistemas de marcação de consultas em consultório médicos, entre outros.

Os sistemas transacionais podem ser organizados de várias formas, porém a mais usual é utilizando-se uma arquitetura cliente-servidor em três camadas. A figura a seguir apresenta essa arquitetura.

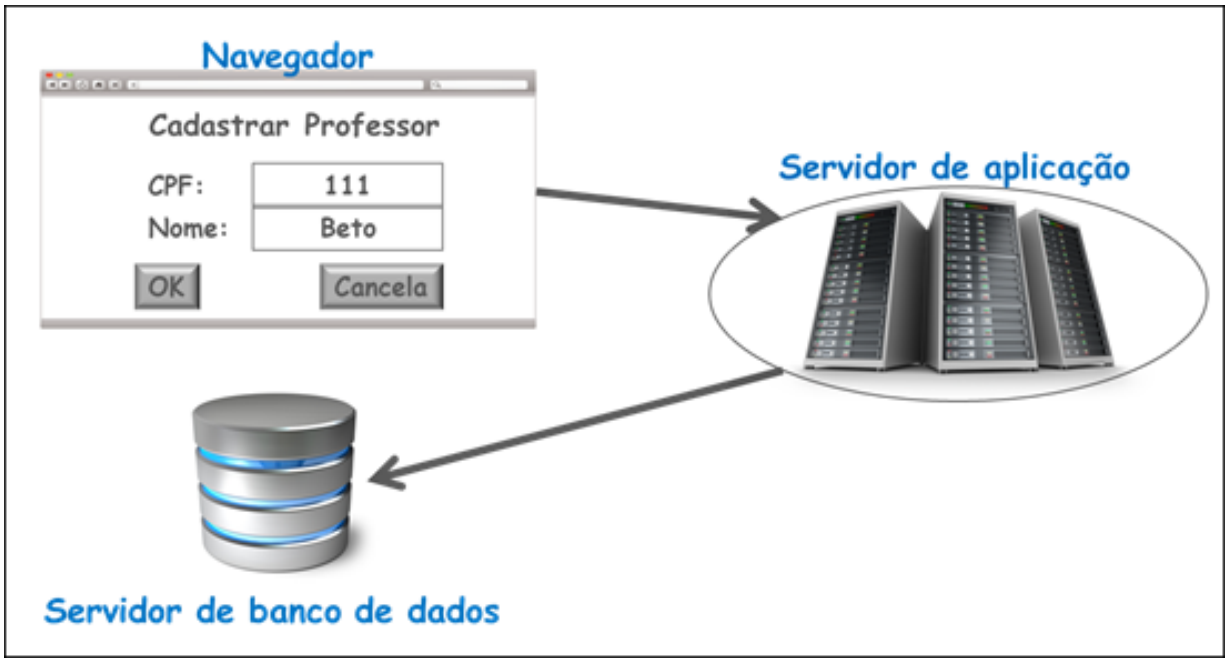


Figura 2.1: Arquitetura cliente-servidor em três camadas.

Conforme apresentado na figura, as camadas de um sistema de informação organizado segundo uma arquitetura cliente-servidor em três camadas são:

- **Camada de apresentação:** software que contém as funcionalidades para visualização dos dados pelos usuários. Normalmente, é composta por um navegador web que apresenta as interfaces que interagirão com os usuários do sistema;

- **Camada da lógica da aplicação:** consiste no software existente no servidor de aplicação, normalmente um servidor web. Esse software implementa as regras do negócio;
- **Camada de armazenamento:** usualmente essa camada é implementada utilizando algum software de armazenamento persistente. Os principais dispositivos de armazenamento (persistência) dos dados são sistemas gerenciadores de banco de dados relacionais. Esse tipo de banco de dados organiza os dados em tabelas.

Para que haja um alinhamento completo entre a TI e o negócio, é necessário que os sistemas transacionais reflitam os processos operacionais da organização. Antes de modelar os sistemas computacionais, deve-se modelar os processos para que esses sistemas sejam desenvolvidos com base nos processos modelados. Neste caso, os processos operacionais são automatizados pelo software da camada de aplicação e os dados gerados e consumidos por esses processos são persistidos pela camada de armazenamento.

Essa afirmação pode parecer óbvia, porém existem organizações que possuem seus processos operacionais modelados, mas os sistemas transacionais são desenvolvidos sem levar em conta os modelos de processos, criando sistemas transacionais totalmente desalinhados aos processos operacionais.

Os dados gerados e armazenados por esses sistemas são de fundamental importância para as organizações, pois eles podem ser reutilizados por outros sistemas, após passarem por processos de transformação.

Um exemplo clássico de sistema transacional é o sistema de vendas de uma empresa. Quando compramos produtos na modalidade on-line de uma empresa, o sistema transacional da loja que acessamos por meio da web registra os nossos dados pessoais, o nosso endereço, a data da compra, os produtos comprados, as

quantidades e os preços dos produtos, entre outros dados, para emitir a nota fiscal.

Esses dados podem ser armazenados em tabelas relacionais em sistemas gerenciadores de banco de dados, o que permite o registro dos dados gerados e consumidos pelos processos operacionais relacionados às vendas dos produtos da empresa. Podemos notar que o objetivo desse sistema é ter o controle pontual de cada venda, permitindo saber quem comprou o quê e quando.

No entanto, esses dados possuem um grande potencial para auxiliar decisões sobre as estratégias de vendas da empresa. Basta criarmos sumarizações desses dados para responder a questões mais abrangentes e chamadas analíticas, por exemplo: "Qual foi o produto mais vendido no ano?", "Qual é o sexo e a faixa etária dos clientes?", "Para quais cidades são feitas mais entregas?", "Qual é o tempo médio das entregas dos produtos?", entre outras.

Cabe ressaltar que esses dados internos dos sistemas transacionais das empresas podem ser enriquecidos com dados externos, como dados de logística de empresas de entrega ou dados demográficos das regiões onde os produtos são vendidos, para proporcionar *insights* mais interessantes.

Os sistemas que proporcionam essas consultas analíticas reutilizam os dados dos sistemas transacionais internos e externos às organizações após transformações e são denominados **sistemas de apoio à decisão** (SAD), que serão tratados na próxima seção.

Sistemas de apoio à decisão

Os **sistemas de apoio à decisão** (SAD) atendem ao nível gerencial da organização, auxiliando gerentes, diretores e presidentes a tomarem decisões que não são facilmente especificadas com antecedência por não serem tarefas estruturadas.

Os SADs utilizam as informações internas geradas pelos sistemas transacionais e frequentemente recorrem a informações de fontes

externas de interesse da organização. Nesses sistemas, as informações internas e externas passam por um processo de extração, transformação e carga para posteriormente serem armazenadas em um repositório dimensional, denominado *Data Warehouse* (DW).

Diferentemente dos sistemas transacionais, que são organizados em uma arquitetura OLTP (*Online Transaction Processing*), por terem como objetivo automatizar processos operacionais, normalmente bem estruturados, os DW têm por objetivo permitir a análise do negócio, por isso são organizados em uma arquitetura OLAP (*Online Analytical Processing*).

Inmon (1997), tido como pioneiro do conceito de Data Warehouse, define DW como uma coleção de dados que tem por objetivo dar suporte aos processos de tomada de decisão. Essa coleção de dados possui as seguintes características:

- **Orientada por assuntos:** refere-se ao fato de o DW armazenar informações sobre temas específicos importantes para o negócio da empresa. Exemplos típicos de temas são: vendas, recursos humanos, clientes, logística etc.;
- **Integrada:** define a representação única para os dados provenientes dos diversos sistemas que formarão a base de dados do DW. A maior parte do trabalho na construção de um DW está na análise dos sistemas transacionais e dos dados que eles contêm. Como não existem padrões de codificação, cada analista pode definir a mesma estrutura de dados de várias formas, fazendo com que dados que retratam a mesma informação sejam representados de diversas maneiras dentro dos sistemas utilizados pela empresa ao longo dos anos. O atributo *sexo*, por exemplo, pode ser representado em um sistema pelas letras "F" para feminino e "M" para masculino, entre outros. Já em outro sistema pode se ter "H" para representar homens e "M" para representar mulheres. Para fazer parte do DW, as informações extraídas dos dois sistemas

devem passar por um processo de transformação para que haja uma interpretação única para o atributo *sexo*;

- **Não volátil:** em um DW, não existem alterações de dados, somente as cargas e as consultas posteriores. Ele é definido assim pois as operações em nível de registro em modo on-line, como são os sistemas transacionais, exigem um controle de consistência muito grande, fugindo do objetivo principal do DW;

- **Variável em relação ao tempo:** todos os dados no DW são precisos em algum instante no tempo. Como eles podem estar corretos somente em um determinado momento, é dito que esses dados "variam com o tempo". Por isso, são necessárias as cargas periódicas para que os dados do DW reflitam o estado presente das organizações.

Kimball (2008), um dos precursores da tecnologia de Data Warehouse, afirma que o DW consiste em todo processo que provê informação para o suporte à tomada de decisão. *Data Warehouse* é, portanto, um repositório de dados corporativo no qual os dados obtidos de sistemas-fontes são devidamente tratados e posteriormente depositados em bancos de dados informacionais, que oferecem um enfoque histórico para permitir um suporte efetivo à decisão. A figura a seguir apresenta a arquitetura de um ambiente de DW proposta por Kimball (2008).

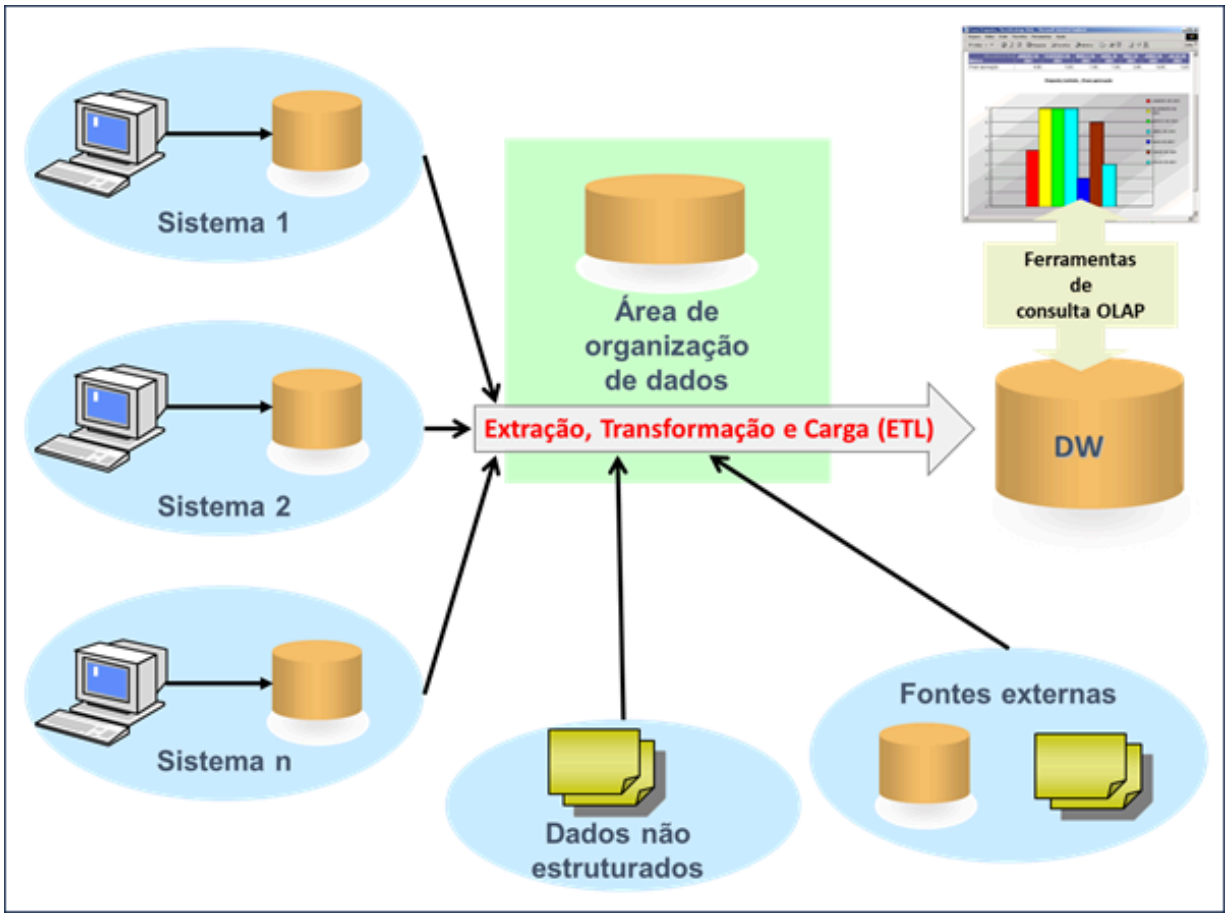


Figura 2.2: Arquitetura de um ambiente de Data Warehouse proposta por Kimball (2008).

A arquitetura apresentada na figura tem como função principal a identificação e o entendimento de como o dado se movimenta, de como é organizado dentro do ambiente de DW e de como será empregado para o fim a que se destina.

Nessa arquitetura, a primeira camada é constituída pelas fontes de dados que correspondem aos dados internos e externos à organização e que serão integradas para compor o DW.

Os dados internos podem ser compostos pelos dados gerados pelos sistemas transacionais — representados, na figura anterior, pelos Sistemas 1 até *n* — ou por qualquer outro dado não estruturado existente na organização, como manuais e regimentos internos. Na grande maioria dos casos, os sistemas transacionais das organizações apresentam-se não integrados, devido ao fato de as

aplicações terem sido construídas uma a uma, separadamente das outras aplicações.

As fontes externas normalmente são heterogêneas e distribuídas, dificultando ainda mais o processo de integração. Cabe ressaltar que essas fontes também podem conter dados não estruturados de interesse do DW, como notícias de jornais, conteúdos de blogs ou de redes sociais.

A área de organização de dados dessa arquitetura basicamente é o local de construção do DW onde são criados os valores a serem adicionados ao DW. A **organização de dados** é um processo complexo que inclui, dentre outros, os seguintes subprocessos: extração, transformação, carga e indexação, verificação da garantia de qualidade, publicação e versionamento, atualização, consultas, auditoria, segurança e cópia de segurança e recuperação.

A segunda camada é a de extração, transformação e carga, denominada ETL (*Extract, Transform, Load*). Essa camada consome em torno de 70% do tempo e esforço de construção de um DW. Nela são realizadas as tarefas de filtragem, transformação e integração dos dados extraídos e são resolvidos vários problemas, como conflitos semânticos e estruturais, diferenças de unidades, diferenças em código ou expressões (KIMBALL, 2008). Após essas atividades, os dados são carregados no DW, a periodicidade da carga deverá ser estabelecida pela regra de negócio e poderá ser diária, semanal, mensal ou qualquer outro período.

Finalmente, a terceira camada é a de persistência representada pelo repositório do DW. Nesse repositório, são conectadas ferramentas de consultas OLAP que normalmente apresentam conjuntos de gráficos (*dashboards*) para a exibição das informações que têm por objetivo subsidiar os gestores em suas decisões.

Conforme apresentado anteriormente, pode-se inferir que criar um DW para uma corporação de grande porte pode se tornar uma tarefa bem difícil, ou até mesmo impossível de se realizar, devido à

complexidade e à quantidade de dados internos estruturados dos sistemas transacionais, dados internos não estruturados e dados externos estruturados e não estruturados a serem tratados para abranger todas as áreas de interesse dessa corporação.

Nesse contexto altamente complexo, surgiu a proposta de Kimball (2008) de criar os *Data Marts* organizados por assuntos para, em seguida, todos os *Data Marts* formarem o DW corporativo. ***Data Mart*** (DM) é um subconjunto lógico de um completo DW. Esse termo é usado para se referir a um DW de pequena capacidade usado para atender especificamente a uma unidade estratégica de negócio ou a um departamento da organização, ou seja, um assunto de interesse da organização. Normalmente, um DM aborda um limitado escopo de dados, projetado para representar uma função particular do negócio, como marketing, vendas ou finanças. A figura a seguir apresenta um exemplo de DW composto por vários DM.

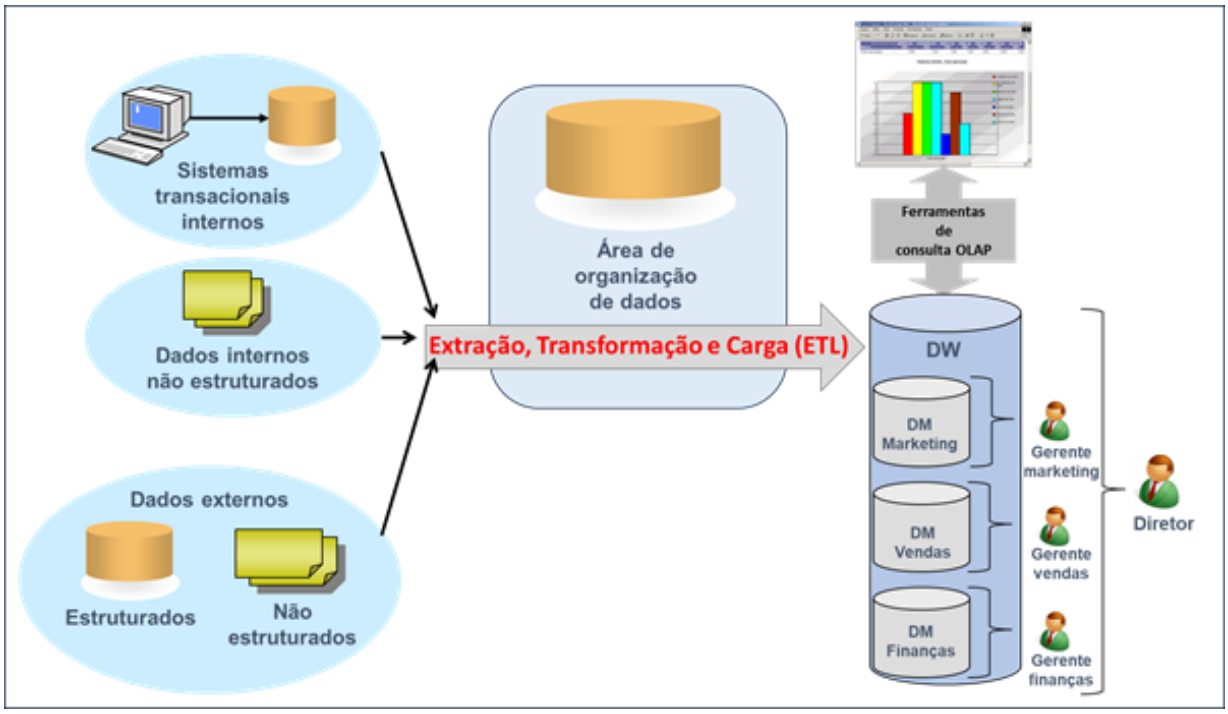


Figura 2.3: Data Warehouse (DW) composto por vários Data Marts (DM).

Como pode ser observado na figura, o DW corporativo é formado pelos DMs de marketing, vendas e finanças. Cabe ressaltar que

existem vários outros assuntos, cada um representado por um DM diferente, a serem abordados em uma organização para a consolidação do seu DW corporativo. Essa abordagem é considerada *iterativa* e *incremental*.

É **iterativa** devido ao fato de a implementação do DW ser organizada em um ciclo de desenvolvimento composto por várias repetições (ou passadas) de um conjunto de procedimentos. Assim, uma solução completa, denominada DW corporativo, não é implementada em apenas uma passada pelo ciclo de desenvolvimento. Na verdade, essa estratégia divide um grande problema, a ser resolvido pelo DW, em pequenos problemas, representados pelos DMs, possibilitando a criação de soluções parciais que comporão a solução final. Cada passada pelo ciclo de desenvolvimento aborda uma área específica da organização gerando um DM. Caso a área seja muito complexa, ela pode possuir mais de uma passada pelo ciclo de desenvolvimento para ser concluída.

É **incremental** devido ao fato de cada passada pelo ciclo de desenvolvimento disponibilizar um incremento, ou seja, uma parte da solução — nesse caso, um DM ou parte dele, que será uma parte da solução que comporá o DW completo.

Essa abordagem possui várias vantagens. Ela possibilita o desenvolvimento de um SAD iniciando por áreas prioritárias para os gestores, além de permitir a entrega de um DM em um tempo menor que o DW completo. Isso faz com que: a pessoa gestora tenha uma visão analítica das áreas mais importantes do negócio em um tempo de espera menor; as equipes de desenvolvimento e de gestão entendam melhor o negócio durante os ciclos de desenvolvimento, possibilitando o incremento da qualidade dos produtos entregues; e o investimento financeiro seja particionado em frações, permitindo uma visão prematura do retorno do investimento.

Após apresentarmos os sistemas transacionais e os sistemas de apoio à decisão, pode-se concluir que esses sistemas possuem objetivos diferentes. Os sistemas transacionais automatizam os processos operacionais das organizações, que são facilmente estruturados, enquanto os sistemas de apoio à decisão são mais complexos por auxiliarem as decisões que dificilmente são estruturadas. Os sistemas transacionais utilizam normalmente modelos relacionais baseados em tabelas compostas por linhas e colunas para armazenar seus dados. Por outro lado, os sistemas de apoio à decisão utilizam modelos dimensionais.

A modelagem de dados desses sistemas foi tratada de forma superficial, pois esses assuntos serão detalhados nos próximos capítulos.

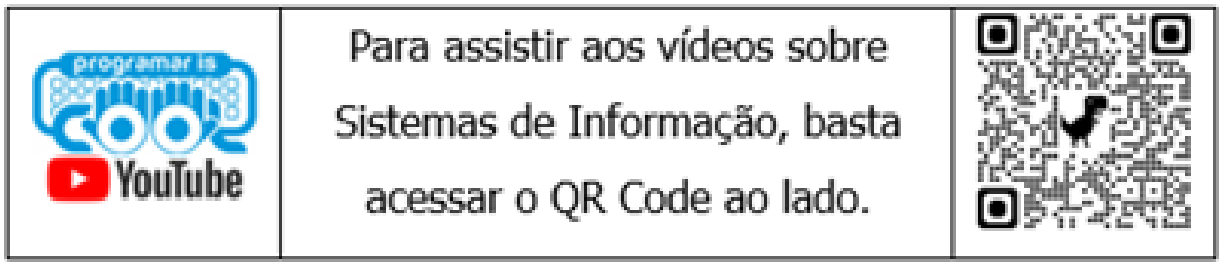


Figura 2.4: Vídeos sobre Sistemas de Informação. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIBcyTSHqkqMVdatnDQzRrUw

Referências

INMON, W. H. *Como Construir o Data Warehouse*. Rio de Janeiro: Campus, 1997.

KIMBALL, R.; ROSS M.; THORNTHWAITE, W.; MUNDY J.; BECKER, B. *The Date Warehouse Lifecycle Toolkit*. 2. Ed. New York: John Wiley & Sons, 2008.

MCGEE, J.; PRUSAK, L. *Gerenciamento Estratégico da Informação*. 11. ed. Rio de Janeiro: Campus, 1994.

ROBREDO, J. *Da Ciência da Informação Revisitada aos Sistemas Humanos de Informação*. Brasília: Thesaurus, 2003.

URDANETA, I. P. *Gestión de la inteligencia, aprendizaje tecnológico y modernización del trabajo informacional: retos y oportunidades*. Caracas: Universidad Simón Bolivar, 1992.

Capítulo 3
Projeto conceitual de banco de dados

Pablo Picasso, renomado artista espanhol, afirmava que "*Em arte, procurar não significa nada. O que importa é encontrar*". Assim como nas artes, no ambiente corporativo, procurar dados não é o suficiente: é necessário *encontrá-los*. Por isso, há a necessidade de se utilizar abordagens sistemáticas para o planejamento da armazenagem dos dados dos sistemas de informação das organizações. Uma das maneiras mais comuns de se planejar é por meio da realização de um projeto de banco de dados para armazená-los em repositórios de dados de forma organizada.

Heuser (2009) afirma que um **banco de dados** (BD) consiste em um conjunto de dados integrados que tem por objetivo atender a uma comunidade de usuários. Por outro lado, segundo o mesmo autor, um **Sistema Gerenciador de Banco de Dados** (SGBD) consiste no software que incorpora as funções de definição, recuperação e alteração de dados em um banco de dados.

Existem vários modelos implementados pelos bancos de dados, sendo que o modelo relacional é o mais popular. Esse modelo foi introduzido por Edgar Frank Codd, pesquisador da IBM, em 1970. Nele, os dados em um banco de dados são representados por uma coleção de tabelas. Cada linha é denominada tupla; uma coluna é chamada de atributo; a tabela é chamada de relação. Sua base teórica reside em uma teoria de conjuntos.

Um projeto de banco de dados relacional possui três níveis de abstração, o conceitual, o lógico e o físico. Neste capítulo, será abordado o projeto conceitual. Os outros níveis de abstração serão apresentados nos próximos capítulos.

3.1 Modelo Entidade-Relacionamento

O objetivo do projeto conceitual é construir o modelo conceitual de dados. A técnica para construir o modelo conceitual de base de dados, denominado modelo **Entidade-Relacionamento** (ER), foi proposta por Peter Chen em 1976 (CHEN, 1990). Esse modelo é representado graficamente pelo Diagrama Entidade-Relacionamento (DER) e é independente de qualquer paradigma ou SGBD específico. O DER é composto pelos seguintes conceitos:

- **Entidade:** conjunto de objetos da realidade modelada, sobre os quais se deseja manter informações na base de dados;
- **Atributo:** dados ou informação que são associados a cada ocorrência de uma entidade ou de um relacionamento;
- **Relacionamento:** conjunto de associações entre entidades;
- **Generalização/Especialização (Herança):** relacionamento que permite atribuir propriedades particulares a um subconjunto das ocorrências especializadas de uma entidade genérica;
- **Cardinalidade de Relacionamento:** a cardinalidade de uma entidade em um relacionamento é o número de ocorrências da entidade associadas à uma ocorrência da entidade conectada por meio do relacionamento.

Entidade e Atributo

Uma **entidade** pode ser entendida como um conjunto de ocorrências da realidade modelada sobre as quais se deseja manter informações na base de dados. É representada graficamente por meio de um retângulo. Esse retângulo contém o nome da entidade e seus **atributos**, que são dados ou informações associadas a cada ocorrência de uma entidade. A figura 3.1 apresenta um exemplo de uma entidade.

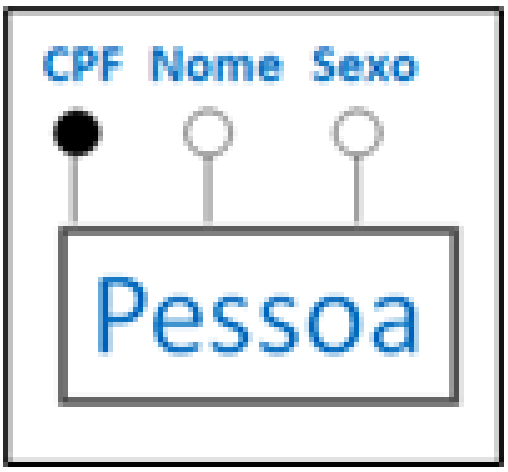


Figura 3.1: Exemplo de entidade.

Como pode ser visto, a entidade `Pessoa` possui os atributos *CPF*, *Nome* e *Sexo*. Pode-se observar que o círculo que representa o atributo CPF está preenchido; isso ocorre devido ao fato de o CPF ser um atributo identificador, ou seja, o atributo que diferencia uma ocorrência de uma entidade das demais. Aqui, estamos utilizando apenas Masculino (M) e Feminino (F) a fim de simplificação do exemplo, embora esta separação binária de gênero mereça uma discussão maior.

Para citar uma ocorrência particular de uma entidade, fala-se em *objeto* ou *instância da entidade*. Exemplos de instâncias ou objetos da entidade `Pessoa` seriam:

- CPF: 111.111.111.11, Nome: Ana, e Sexo: F;
- CPF: 222.222.222-22, Nome: Beto, e Sexo: M;
- CPF: 333.333.333-33, Nome: Caio, e Sexo: M;
- CPF: 444.444.444-44, Nome: Beto, e Sexo: M;

Dados os exemplos de instâncias da entidade `Pessoa`, pode-se observar que as pessoas com o CPF 222.222.222-22 e 444.444.444-44 possuem o mesmo nome e sexo, porém, representam instâncias distintas, pois o CPF garante essa existência única por se tratar de um atributo identificador. Trata-se de um exemplo de pessoas

homônimas. Toda instância de uma entidade deve ser diferente das demais, esse é o princípio da unicidade.

Relacionamento

Um **relacionamento** representa um conjunto de associações entre entidades. A figura 3.2 apresenta um exemplo de relacionamento.

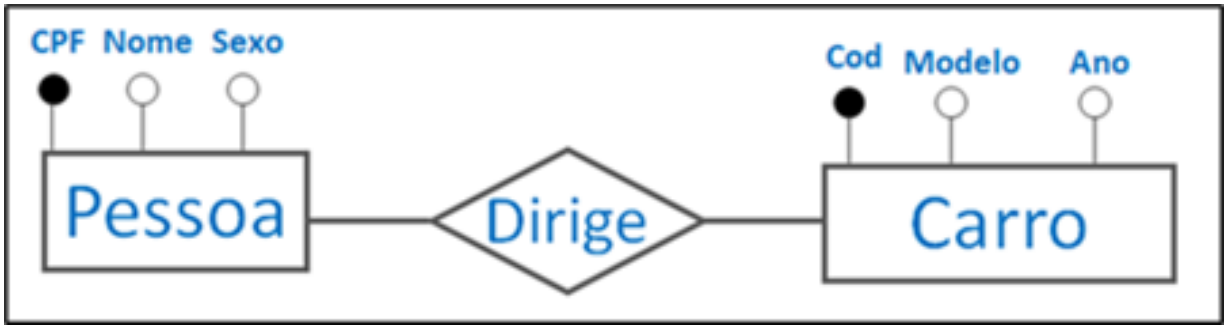


Figura 3.2: Exemplo de relacionamento entre entidades.

A figura 3.2 apresenta o relacionamento entre duas entidades, `Pessoa` e `Carro`. Denominado `Dirige`, esse relacionamento é representado por um losango. Assim, uma instância de `Pessoa`, cujos dados estejam armazenados no banco de dados, poderá se relacionar com uma instância de `Carro`, cujos dados também estejam no banco de dados, por meio do relacionamento `Dirige`. Esse modelo nos passa as seguintes informações:

- Existência de um conjunto de instâncias de `Pessoas`;
- Existência de um conjunto de instâncias de `Carros`;
- Existência de um conjunto de associações que ligam `Pessoas` a `Carros`.

Generalização/Especialização (Herança)

A **Generalização/Especialização**, também denominada **Herança**, permite atribuir propriedades particulares a um subconjunto das ocorrências especializadas de uma entidade

genérica. A figura 3.3 apresenta um exemplo de Generalização/Especialização.

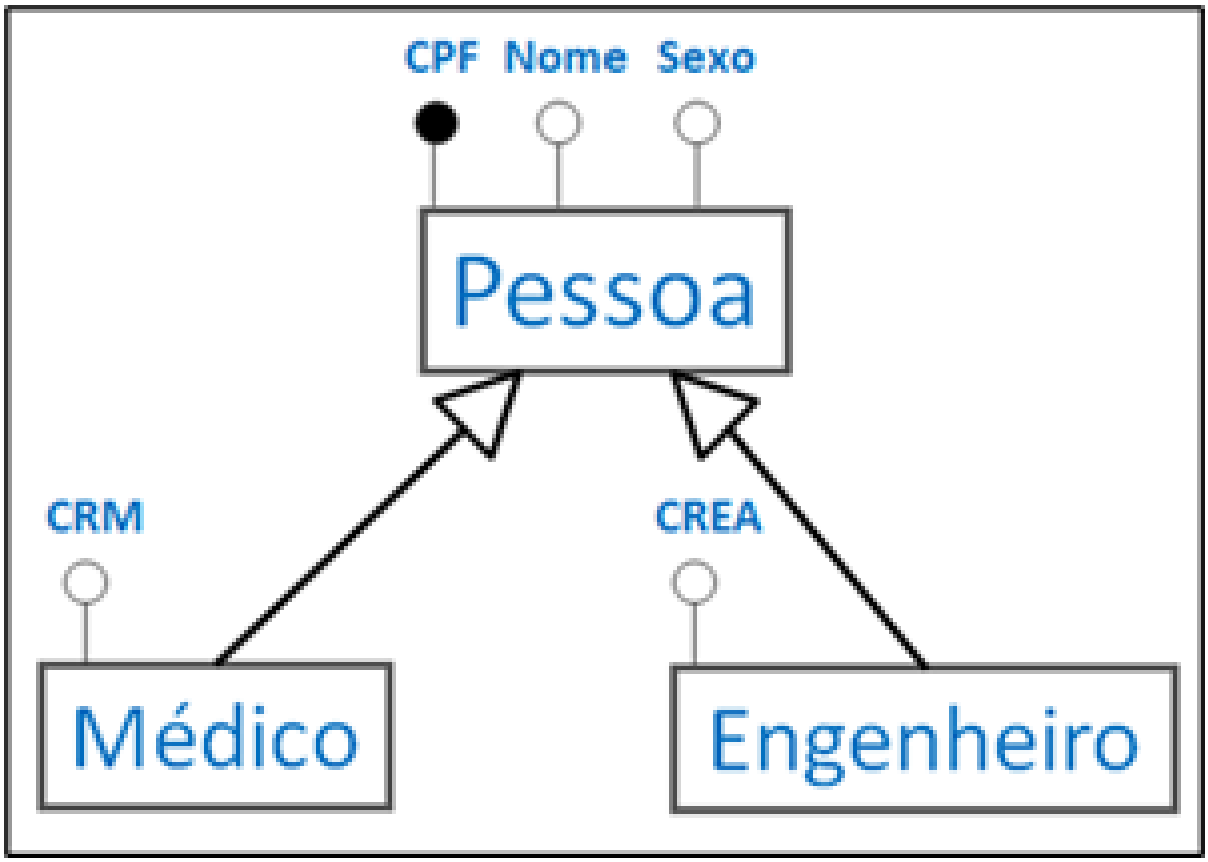


Figura 3.3: Exemplo de Generalização/Especialização ou Herança.

Nessa figura, a entidade `Pessoa` é a entidade genérica; já as entidades `Médico` e `Engenheiro` são as entidades especializadas, ou seja, são as que herdam os atributos da entidade genérica `Pessoa`. A leitura do modelo seria "Médico é um tipo de Pessoa" e "Engenheiro é um tipo de Pessoa".

Ao criarmos uma instância da entidade `Pessoa`, ela possuirá os atributos *CPF*, *Nome* e *Sexo*. Por outro lado, ao criarmos uma instância da entidade `Médico`, essa instância possuirá os atributos *CPF*, *Nome* e *Sexo*, herdados da entidade genérica `Pessoa`, mais o atributo *CRM*, exclusivo da entidade especializada `Médico`, assim como, ao criarmos uma instância da entidade `Engenheiro`, essa instância possuirá os mesmos atributos herdados da entidade

genérica `Pessoa`, mais o atributo *CREA*, exclusivo da entidade especializada `Engenheiro`.

Um exemplo de instância ou objeto da entidade `Médico` seria:

- CPF: 777.777.777-77, Nome: Edu, Sexo: M e CRM: M777.

Um exemplo de instância ou objeto da entidade `Engenheiro` seria:

- CPF: 888.888.888-88, Nome: Lia, Sexo: F e CREA: E888.

Cardinalidades de relacionamentos

Para definir o número de ocorrências de uma entidade em um relacionamento, usamos o conceito de cardinalidade. A **cardinalidade** indica quantas ocorrências de uma entidade participam no mínimo e no máximo do relacionamento. A cardinalidade mínima define se o relacionamento entre duas entidades é obrigatório ou não. Por outro lado, a cardinalidade máxima define a quantidade máxima de ocorrências da entidade que podem participar do relacionamento, por isso sempre deve ser maior que 0 (zero). As cardinalidades de relacionamentos binários são: 1:1, 1:n e n:n.

Cardinalidade 1:1

Em um relacionamento com cardinalidade 1:1, uma instância de uma entidade pode se relacionar com no máximo uma instância da outra entidade. Essa sintaxe, representada por meio de dois pontos entre as cardinalidades de duas entidades, apresenta apenas as cardinalidades máximas; as cardinalidades mínimas são representadas no DER. A figura 3.4 apresenta um exemplo de relacionamento com cardinalidade 1:1.

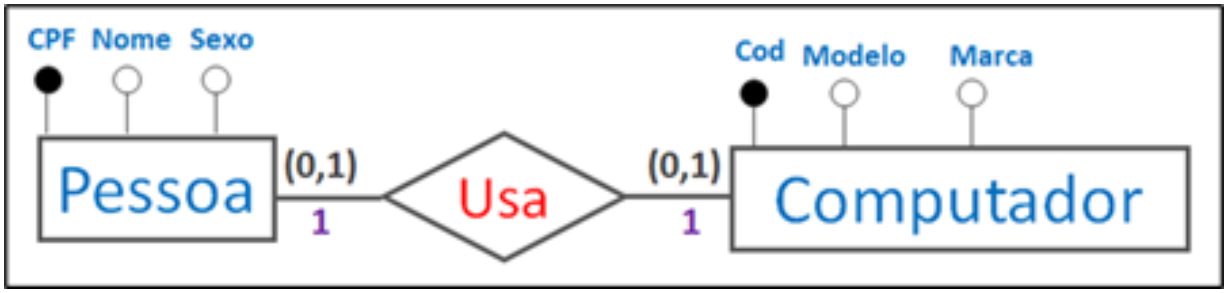


Figura 3.4: Exemplo de DER com relacionamento com cardinalidade 1:1.

A figura 3.4 apresenta um DER com o relacionamento `Usa` entre as entidades `Pessoa` e `Computador`. A cardinalidade desse relacionamento é 1:1, e isso significa que uma instância de `Pessoa` pode estar relacionada no máximo a uma instância de `Computador`, assim como uma instância de `Computador` pode estar relacionada no máximo a uma instância de `Pessoa`.

As cardinalidades mínima e máxima estão representadas entre os parênteses próximos aos retângulos que representam as entidades. Nesse caso, a cardinalidade mínima e máxima do relacionamento são 0 e 1, respectivamente, em ambos os lados. Isso significa que uma instância de `Pessoa` pode estar relacionada, no máximo, a uma e, no mínimo, a nenhuma instância de `Computador`; e uma instância de `Computador` pode estar relacionada, no máximo, a uma e, no mínimo, a nenhuma instância de `Pessoa`.

Quando a cardinalidade mínima é 0, dizemos que o relacionamento é *opcional*; já quando a cardinalidade mínima é 1, dizemos que o relacionamento é obrigatório. A figura 3.5 apresenta esse relacionamento por meio da teoria de conjuntos.

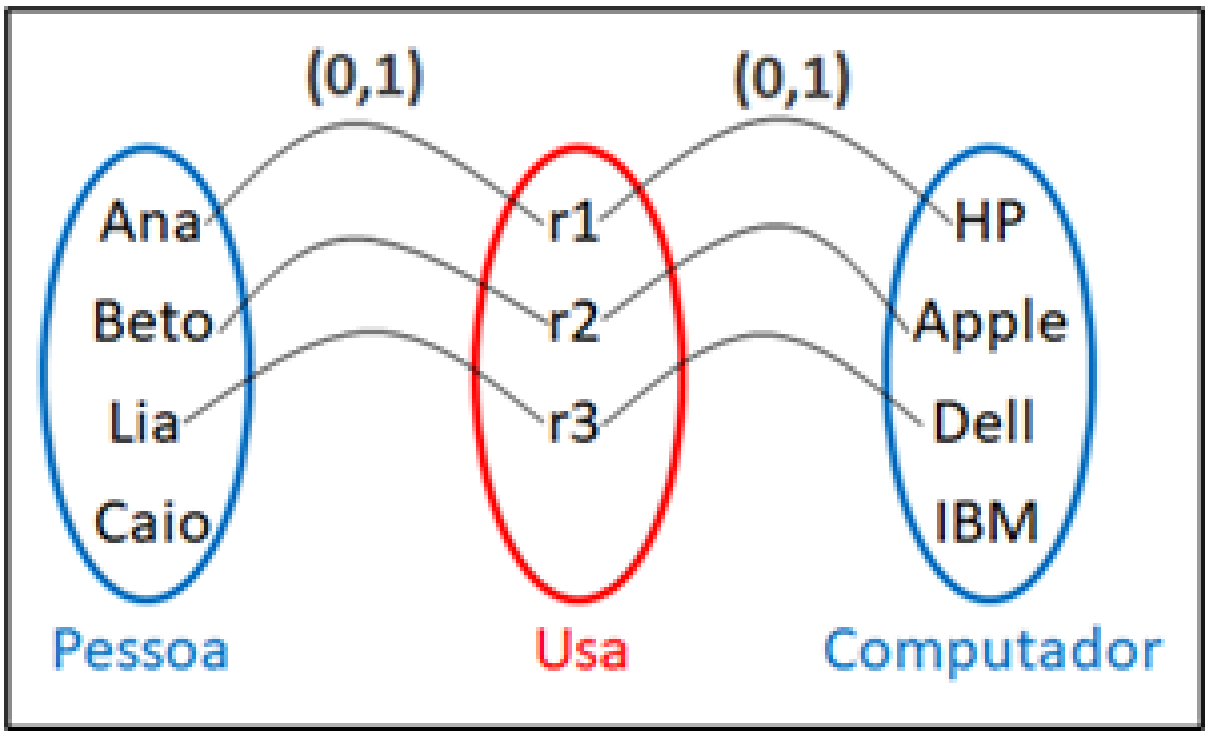


Figura 3.5: Exemplo de conjuntos representando relacionamento com cardinalidade 1:1.

Como pode ser observado na figura 3.5, a entidade `Pessoa` se relaciona com a entidade `Computador` por meio do relacionamento `Usa`. As entidades `Pessoa` e `Computador` e o relacionamento `Usa` são representados por conjuntos com os respectivos elementos. As instâncias de `Pessoa`, ou elementos do conjunto `Pessoa`, "Ana", "Beto" e "Lia", se relacionam no máximo com uma instância da entidade `Computador` (ou elemento do conjunto `Computador`) por meio das instâncias "r1", "r2" e "r3" do relacionamento `Usa` (ou elementos do conjunto `Usa`). No entanto, a instância "Caio" da entidade `Pessoa` não se relaciona com nenhuma instância de `Computador` e a instância "IBM" de `Computador` não se relaciona com nenhuma instância de `Pessoa`.

Conforme apresentado, pode-se concluir que a cardinalidade máxima igual a 1 ocorre quando um elemento de um conjunto se relaciona no máximo com um elemento de outro conjunto, e a cardinalidade mínima igual a 0 ocorre quando um elemento de um conjunto não se relaciona com nenhum elemento de outro conjunto,

ou seja, esse elemento não participa do relacionamento, tornando-o opcional.

Existem outras formas de cardinalidades mínimas em um relacionamento com cardinalidade 1:1, pois a cardinalidade máxima já está estabelecida com o valor 1 na sintaxe. As figuras 3.6 e 3.8 apresentam outras possibilidades de cardinalidades mínimas em relacionamentos com cardinalidade 1:1. A figura 3.6 apresenta o relacionamento obrigatório em uma das entidades participantes do relacionamento e a figura 3.8 apresenta o relacionamento obrigatório em ambas as entidades.

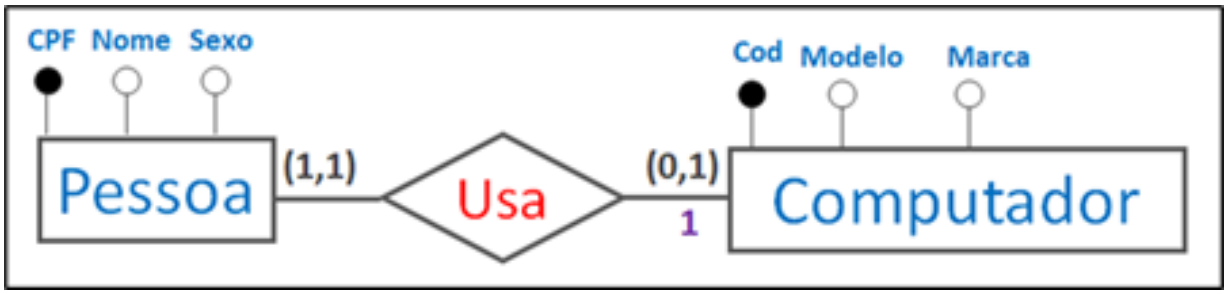


Figura 3.6: Exemplo de DER com um relacionamento obrigatório e cardinalidade 1:1.

A figura 3.6 apresenta o mesmo DER da figura 3.4, porém a figura 3.6 apresenta um exemplo de relacionamento obrigatório por parte de uma das entidades, diferentemente da figura 3.4, na qual o relacionamento é opcional em ambas as entidades. Na figura 3.6, a cardinalidade mínima e máxima do relacionamento são 1 e 1, no lado esquerdo, e 0 e 1, no lado direito. Isso significa que uma instância de `Pessoa` pode estar relacionada, no mínimo, a nenhuma e, no máximo, a uma instância de `Computador` ; e uma instância de `Computador` deve estar relacionada, no mínimo, a uma e, no máximo, a uma instância de `Pessoa` .

Quando a cardinalidade mínima é 0, dizemos que o relacionamento é opcional; já quando a cardinalidade mínima é 1, dizemos que o relacionamento é obrigatório. A figura 3.7 apresenta esse relacionamento por meio da teoria de conjuntos.

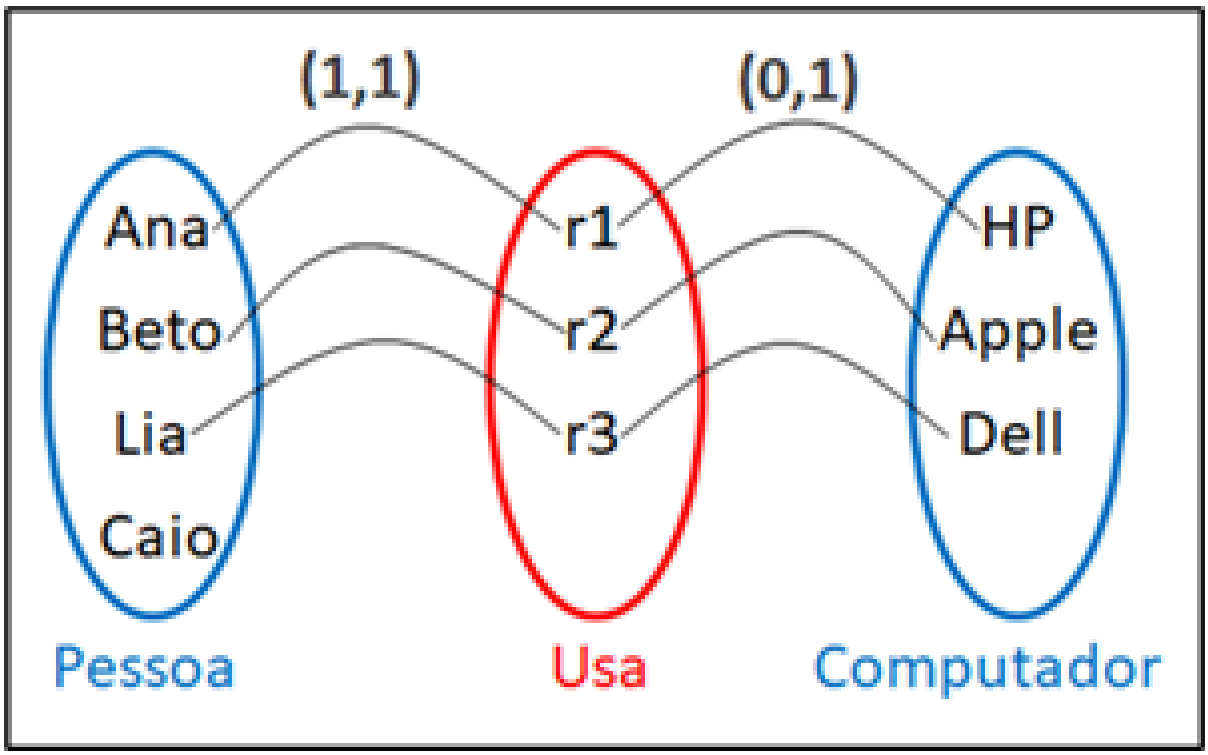


Figura 3.7: Exemplo de conjuntos representando o relacionamento com cardinalidade 1:1 com participação obrigatória de uma das entidades.

Como pode ser observado na figura 3.7, as instâncias de `Pessoa`, ou elementos do conjunto `Pessoa`, "Ana", "Beto" e "Lia", se relacionam no máximo com uma instância da entidade `Computador` (ou elemento do conjunto `Computador`) por meio das instâncias "r1", "r2" e "r3" do relacionamento `Usa` (ou elementos do conjunto `Usa`). No entanto, a instância "Caio" da entidade `Pessoa` não se relaciona com nenhuma instância de `Computador`. Por outro lado, todas as instâncias de `Computador` se relacionam com exatamente uma instância de `Pessoa`, o que caracteriza um relacionamento obrigatório — ou seja, para inserir uma instância de `Computador`, será necessário que este novo elemento se relacione a uma instância de `Pessoa`.

Cabe ressaltar que o relacionamento também pode ser obrigatório para as duas entidades. Nesse caso, para inserir uma instância de `Computador`, é necessário que ela se relacione a uma instância de `Pessoa` e, para inserir uma instância de `Pessoa`, é necessário que ela

se relacione a uma instância de `Computador`, conforme apresenta a figura 3.8.

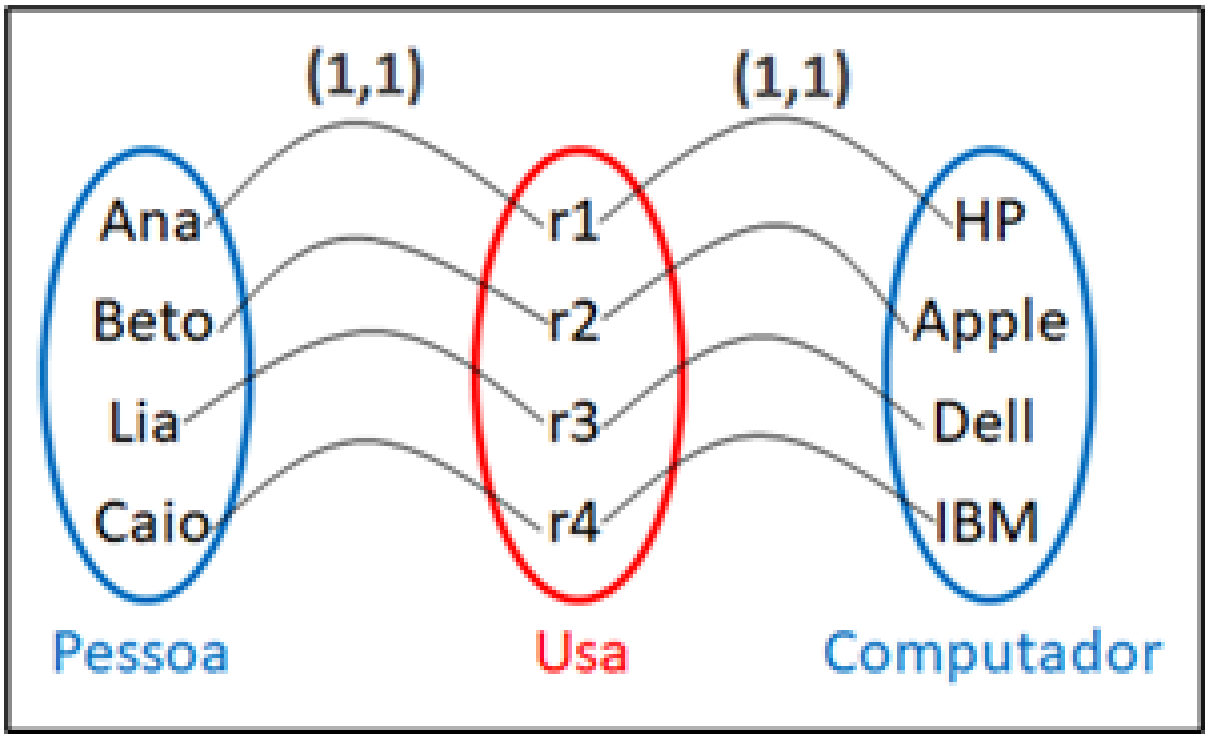


Figura 3.8: Exemplo de conjuntos representando o relacionamento com cardinalidade 1:1 com participação obrigatória de ambas as entidades.

Como pode ser observado na figura 3.8, cada instância de `Computador` se relaciona com exatamente uma instância de `Pessoa` e vice-versa, ou seja, não há elemento em um dos dois conjuntos que não esteja relacionado a exatamente um elemento do outro conjunto.

Cardinalidade 1:n

Em um relacionamento com cardinalidade 1:n, uma instância de uma entidade pode se relacionar com, no máximo, várias instâncias da outra entidade. Essa sintaxe, representada por meio de dois pontos entre as cardinalidades de duas entidades, apresenta apenas as cardinalidades máximas; as cardinalidades mínimas são representadas no DER. A figura 3.9 apresenta um exemplo de relacionamento com cardinalidade 1:n.

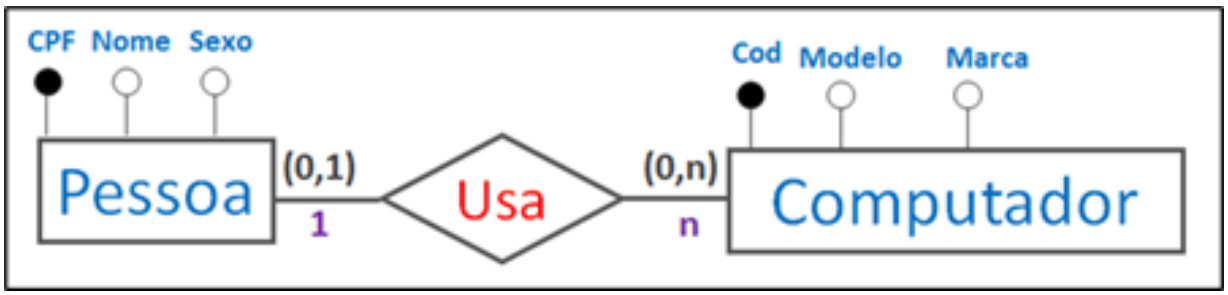


Figura 3.9: Exemplo de DER com relacionamento com cardinalidade 1:n.

A figura 3.9 apresenta o DER com o relacionamento `Usa` entre as entidades `Pessoa` e `Computador`. A cardinalidade desse relacionamento é *1:n*, o que significa que uma instância de `Pessoa` pode estar relacionada, no máximo, a várias (*n*) instâncias de `Computador`, e uma instância de `Computador` pode estar relacionada, no máximo, a 1 instância de `Pessoa`.

Nesse caso, as cardinalidades mínima e máxima do relacionamento são 0 e 1, do lado esquerdo do diagrama, e 0 e *n*, do lado direito. Isso significa que uma instância de `Pessoa` pode estar relacionada no máximo a várias e, no mínimo, a nenhuma instância de `Computador`; e uma instância de `Computador` pode estar relacionada no máximo a uma e, no mínimo, a nenhuma instância de `Pessoa`. A figura 3.10 apresenta esse relacionamento por meio da teoria de conjuntos.

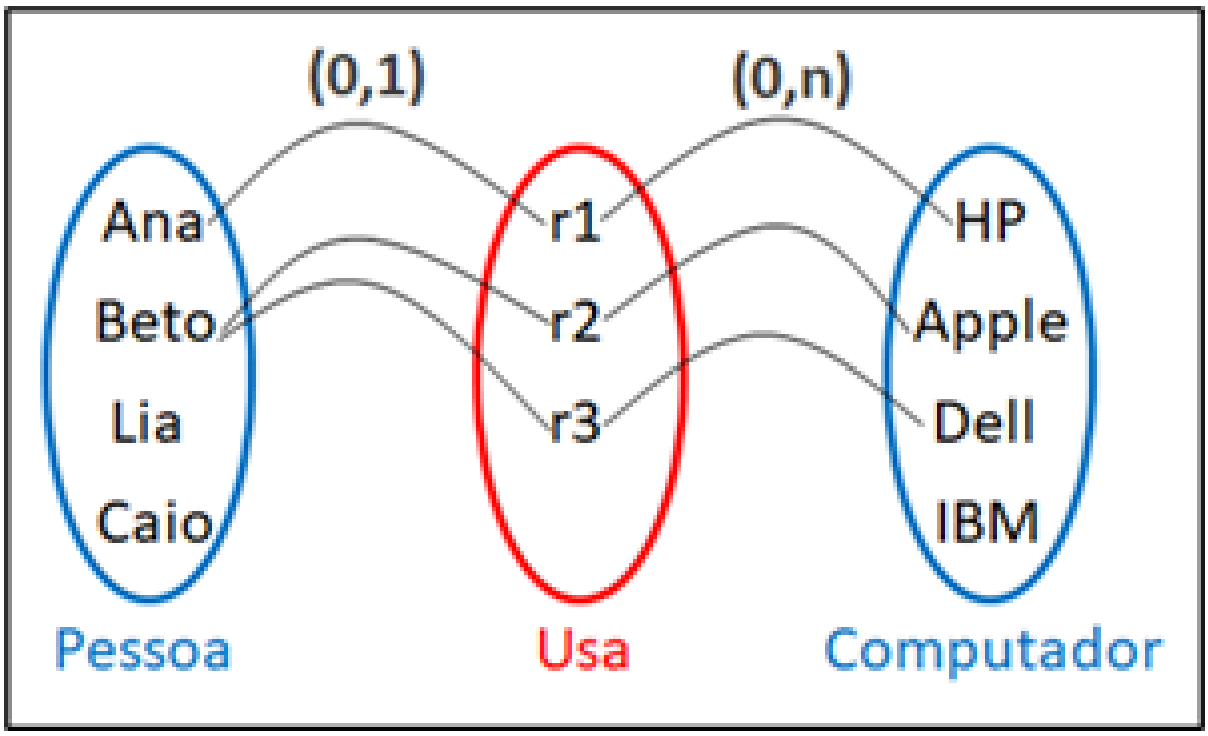


Figura 3.10: Exemplo de conjuntos representando relacionamento com cardinalidade 1:n.

Como pode ser observado na figura 3.10, a instância "Ana" de `Pessoa` se relaciona com a instância "HP" da entidade `Computador` por meio da instância "r1" do relacionamento `Usa`. A instância "Beto" de `Pessoa` se relaciona com as instâncias "Apple" e "Dell" da entidade `Computador` por meio das instâncias "r2" e "r3" do relacionamento `Usa`. As instâncias "Lia" e "Caio" da entidade `Pessoa` não se relacionam com nenhuma instância de `Computador`. Por outro lado, as instâncias "HP", "Apple" e "Dell" da entidade `Computador` se relacionam com apenas uma instância da entidade `Pessoa`, e a instância "IBM" de `Computador` não se relaciona com nenhuma instância de `Pessoa`.

Tal como no relacionamento de cardinalidade 1:1, também existem outras formas de cardinalidades mínimas em um relacionamento com cardinalidade 1:n, apresentadas nas próximas três figuras. A figura 3.11 apresenta o relacionamento obrigatório da entidade do lado esquerdo do relacionamento; a figura 3.12 apresenta o relacionamento obrigatório da entidade do lado direito do

relacionamento e a figura 3.13 apresenta o relacionamento obrigatório em ambas as entidades.

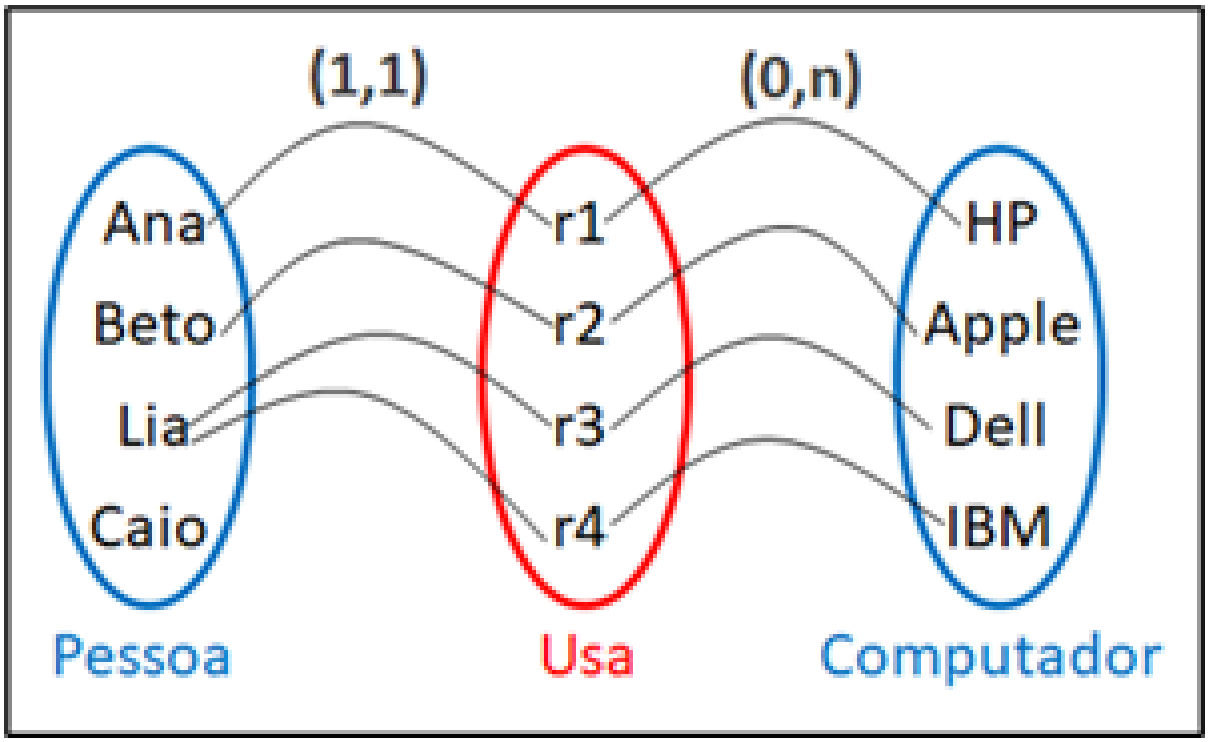


Figura 3.11: Exemplo de conjuntos representando um relacionamento obrigatório com cardinalidade 1:n.

Como pode ser observado na figura 3.11, as instâncias "Ana" e "Beto" de `Pessoa` se relacionam com as instâncias "HP" e "Apple" da entidade `Computador` por meio das instâncias "r1" e "r2", respectivamente, do relacionamento `Usa`. A instância "Lia" de `Pessoa` se relaciona com as instâncias "Dell" e "IBM" da entidade `Computador` por meio das instâncias "r3" e "r4" do relacionamento `Usa`. Finalmente, a instância "Caio" da entidade `Pessoa` não se relaciona com nenhuma instância de `Computador`. Por outro lado, cada instância da entidade `Computador` se relaciona com uma instância da entidade `Pessoa`, devido ao fato de a cardinalidade mínima ser 1 — ou seja, para inserir uma instância na entidade `Computador`, será necessário que ela se relacione com uma instância da entidade `Pessoa`.

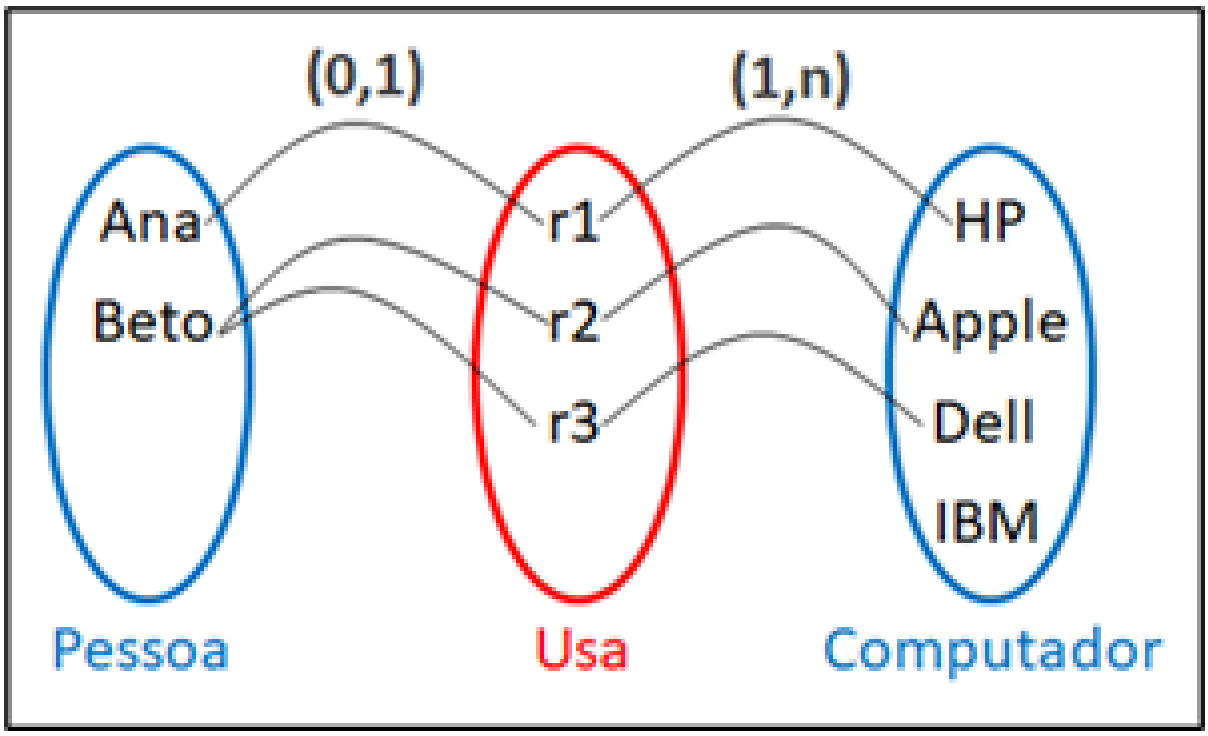


Figura 3.12: Exemplo de conjuntos representando um relacionamento obrigatório com cardinalidade 1:n.

Como pode ser observado na figura 3.12, a instância "Ana" de `Pessoa` se relaciona com a instância "HP" da entidade `Computador` por meio da instância "r1" do relacionamento `Usa`. A instância "Beto" de `Pessoa` se relaciona com as instâncias "Apple" e "Dell" da entidade `Computador` por meio das instâncias "r2" e "r3" do relacionamento `Usa`. Por outro lado, as instâncias "HP", "Apple" e "Dell" da entidade `Computador` se relacionam com apenas uma instância da entidade `Pessoa`, e a instância "IBM" de `Computador` não se relaciona com nenhuma instância de `Pessoa`. Nesse caso, para inserir uma instância na entidade `Pessoa`, será necessário que ela se relacione com uma instância da entidade `Computador`.

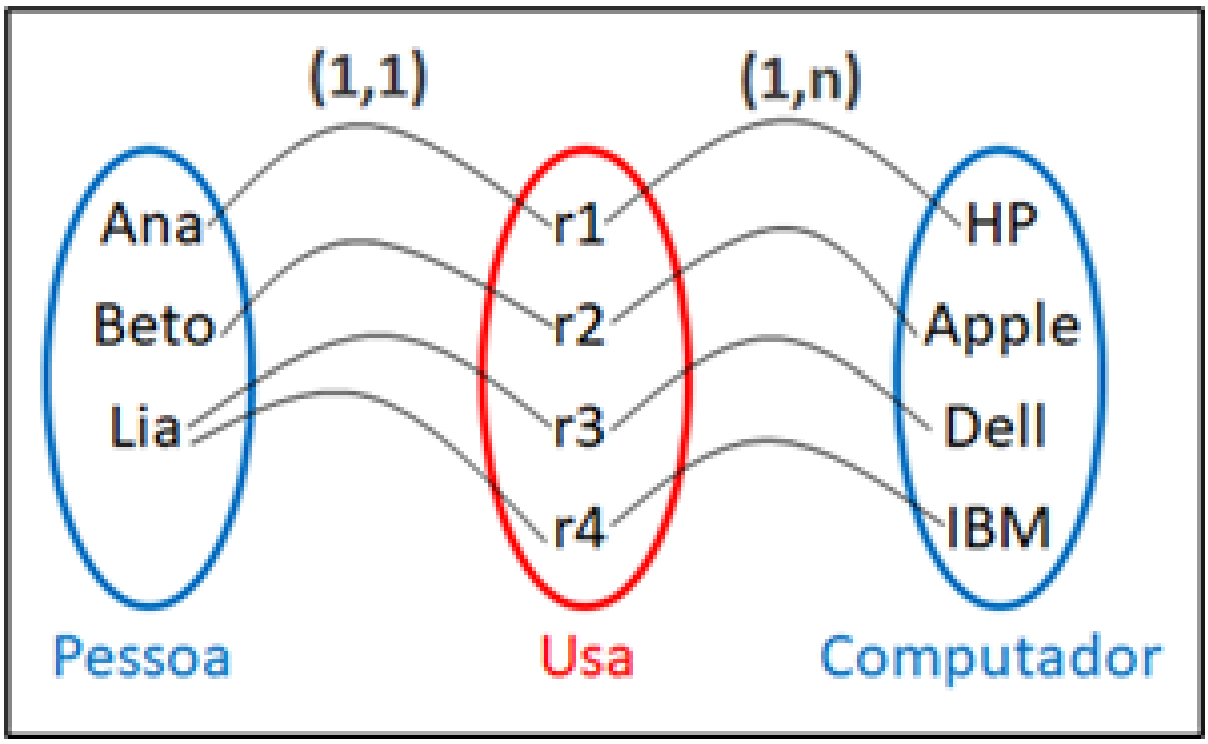


Figura 3.13: Exemplo de conjuntos representando o relacionamento com cardinalidade 1:n com participação obrigatória de ambas as entidades.

Como pode ser observado na figura 3.13, as instâncias "Ana" e "Beto" de `Pessoa` se relacionam com as instâncias "HP" e "Apple" da entidade `Computador` por meio das instâncias "r1" e "r2", respectivamente, do relacionamento `Usa`. A instância "Lia" de `Pessoa` se relaciona com as instâncias "Dell" e "IBM" da entidade `Computador` por meio das instâncias "r3" e "r4" do relacionamento `Usa`. Por outro lado, cada instância da entidade `Computador` se relaciona com uma instância da entidade `Pessoa`. Devido ao fato de a cardinalidade mínima ser 1 em ambos os lados do relacionamento, para inserir uma instância na entidade `Computador`, será necessário que ela se relacione com uma instância da entidade `Pessoa` e, para inserir uma instância na entidade `Pessoa`, será necessário que ela se relacione com uma instância da entidade `Computador`.

Cardinalidade n:n

Em um relacionamento com cardinalidade **n:n**, uma instância de uma entidade pode se relacionar com, no máximo, várias instâncias

da outra entidade em ambos os lados do relacionamento. Essa sintaxe, representada por meio de dois pontos entre as cardinalidades de duas entidades, apresenta apenas as cardinalidades máximas; as cardinalidades mínimas são representadas no DER. A figura 3.14 apresenta um exemplo de relacionamento com cardinalidade n:n.

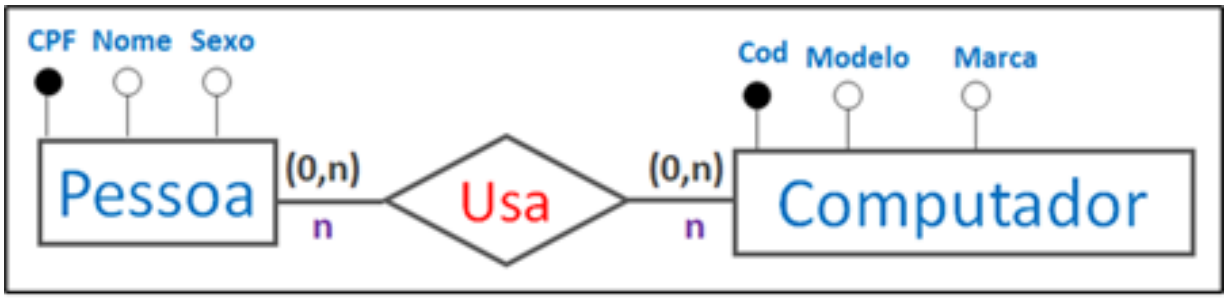


Figura 3.14: Exemplo de DER com relacionamento com cardinalidade n:n.

A figura 3.14 apresenta o DER com o relacionamento `Usa` entre as entidades `Pessoa` e `Computador`. A cardinalidade desse relacionamento é *n:n*; isso significa que uma instância de `Pessoa` pode estar relacionada no máximo a várias (*n*) instâncias de `Computador`, e uma instância de `Computador` pode estar relacionada no máximo a várias (*n*) instâncias de `Pessoa`.

Nesse caso, as cardinalidades mínima e máxima do relacionamento são 0 e *n* em ambos os lados. Isso significa que uma instância de `Pessoa` pode estar relacionada, no máximo, a várias e, no mínimo, a nenhuma instância de `Computador`; e uma instância de `Computador` pode estar relacionada, no máximo, a várias e, no mínimo, a nenhuma instância de `Pessoa`. A figura 3.15 apresenta esse relacionamento por meio da teoria de conjuntos.

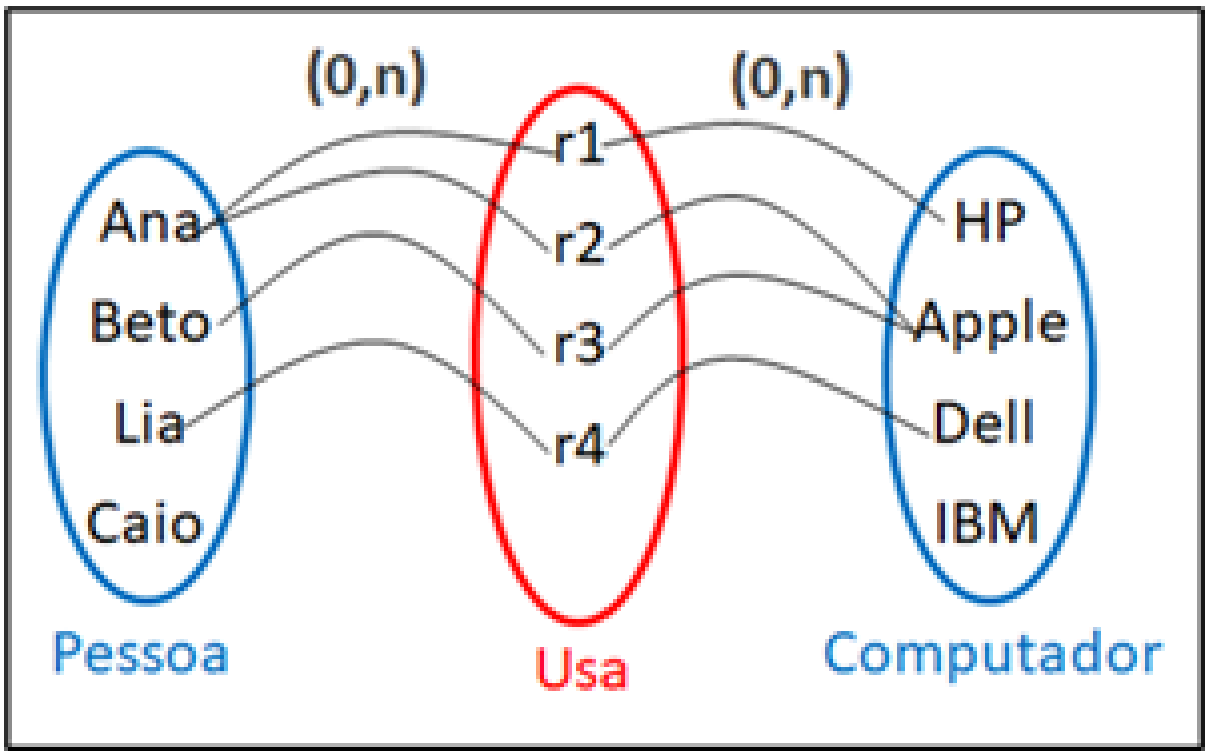


Figura 3.15: Exemplo de conjuntos representando relacionamento com cardinalidade n:n.

Como pode ser observado na figura 3.15, a instância "Ana" de `Pessoa` se relaciona com as instâncias "HP" e "Apple" da entidade `Computador` por meio das instâncias "r1" e "r2" do relacionamento `Usa`. A instância "Beto" de `Pessoa` se relaciona com a instância "Apple" da entidade `Computador` por meio da instância "r3" do relacionamento `Usa`. A instância "Lia" da entidade `Pessoa` se relaciona com a instância "Dell" da entidade `Computador` por meio da instância "r4". Finalmente, a instância "Caio" da entidade `Pessoa` não se relaciona com nenhuma instância de `Computador`.

Por outro lado, a instância "HP" de `Computador` se relaciona com a instância "Ana" de `Pessoa` por meio do relacionamento "r1" de `Usa`. A instância "Apple" de `Computador` se relaciona com as instâncias "Ana" e "Beto" de `Pessoa` por meio dos relacionamentos "r2" e "r3" de `Usa`. A instância "Dell" de `Computador` se relaciona com a instância "Lia" de `Pessoa` por meio do relacionamento "r4" de `Usa`. Finalmente, a instância "IBM" de `Computador` não se relaciona com nenhuma instância de `Pessoa`.

Tal como nos relacionamentos de cardinalidades 1:1 e 1:n, também existem outras formas de cardinalidades mínimas em um relacionamento com cardinalidade n:n. A figura 3.16 apresenta o relacionamento obrigatório das entidades em um dos lados do relacionamento, pois a lógica é a mesma para os dois lados. A figura 3.17 apresenta o relacionamento obrigatório em ambas as entidades.

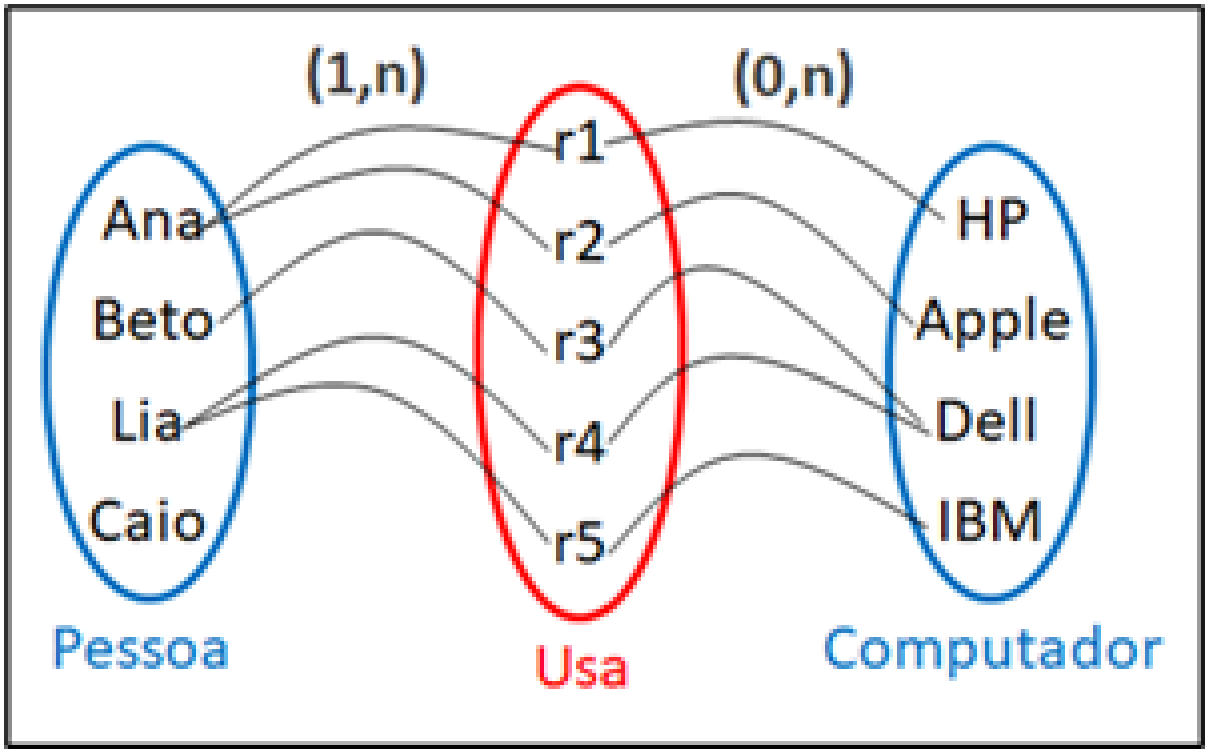


Figura 3.16: Exemplo de conjuntos representando um relacionamento obrigatório com cardinalidade n:n.

Como pode ser observado na figura 3.16, a instância "Ana" de `Pessoa` se relaciona com as instâncias "HP" e "Apple" da entidade `Computador` por meio das instâncias "r1" e "r2" do relacionamento `Usa`. A instância "Beto" de `Pessoa` se relaciona com a instância "Dell" da entidade `Computador` por meio da instância "r3" do relacionamento `Usa`. A instância "Lia" da entidade `Pessoa` se relaciona com as instâncias "Dell" e "IBM" da entidade `Computador` por meio das instâncias "r4" e "r5". Finalmente, a instância "Caio" da entidade `Pessoa` não se relaciona com nenhuma instância de `Computador`.

Por outro lado, as instâncias “HP” e “Apple” de `Computador` se relacionam com a instância “Ana” de `Pessoa` por meio dos relacionamentos “r1” e “r2” de `Usa`. A instância “Dell” de `Computador` se relaciona com as instâncias “Beto” e “Lia” de `Pessoa` por meio dos relacionamentos “r3” e “r4” de `Usa`. Finalmente, a instância “IBM” de `Computador` se relaciona com a instância “Lia” de `Pessoa` por meio do relacionamento “r5” de `Usa`.

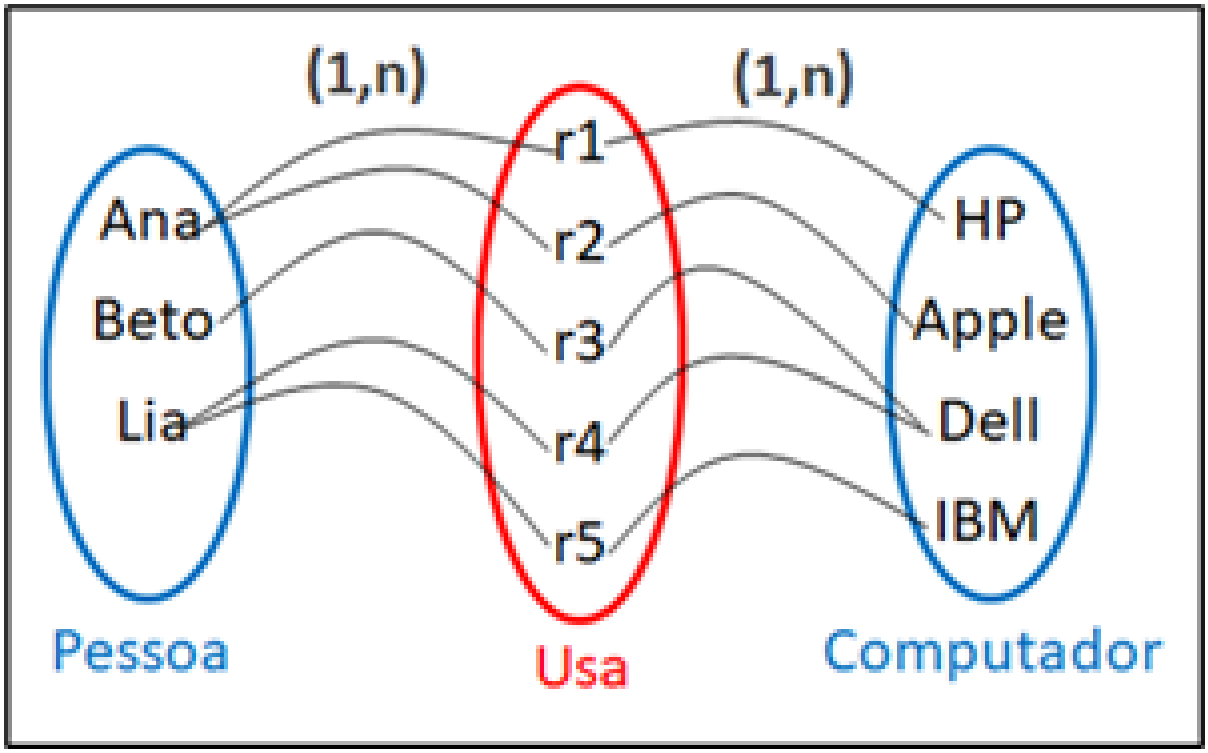


Figura 3.17: Exemplo de conjuntos representando o relacionamento com cardinalidade n:n com participação obrigatória de ambas as entidades.

Como pode ser observado na figura 3.17, a instância “Ana” de `Pessoa` se relaciona com as instâncias “HP” e “Apple” da entidade `Computador` por meio das instâncias “r1” e “r2” do relacionamento `Usa`. A instância “Beto” de `Pessoa` se relaciona com a instância “Dell” da entidade `Computador` por meio da instância “r3” do relacionamento `Usa`. A instância “Lia” da entidade `Pessoa` se relaciona com as instâncias “Dell” e “IBM” da entidade `Computador` por meio das instâncias “r4” e “r5”.

Por outro lado, as instâncias “HP” e “Apple” de `Computador` se relacionam com a instância “Ana” de `Pessoa` por meio dos relacionamentos “r1” e “r2” de `Usa`. A instância “Dell” de `Computador` se relaciona com as instâncias “Beto” e “Lia” de `Pessoa` por meio dos relacionamentos “r3” e “r4” de `Usa`. Finalmente, a instância “IBM” de `Computador` se relaciona com a instância “Lia” de `Pessoa` por meio do relacionamento “r5” de `Usa`.

3.2 Notação da engenharia da informação

Na seção anterior, nós abordamos o DER proposto por Peter Chen, porém existem outras notações para representar o modelo conceitual, dentre elas a **notação da engenharia da informação**, proposta por James Martin (1991) no final da década de 1980.

Na verdade, a notação da engenharia da informação é a mais utilizada por ferramentas CASE (*Computer-Aided Software Engineering*), que são ferramentas baseadas em computadores que auxiliam as atividades de engenharia de software, ou seja, são softwares utilizados para gerar modelos, códigos de programas, testes de códigos etc.

A notação da engenharia da informação é muito semelhante à notação do DER. As principais diferenças são a maneira de representar os relacionamentos e suas cardinalidades. A figura 3.18 apresenta como as cardinalidades dos relacionamentos são representadas utilizando a notação da engenharia da informação.

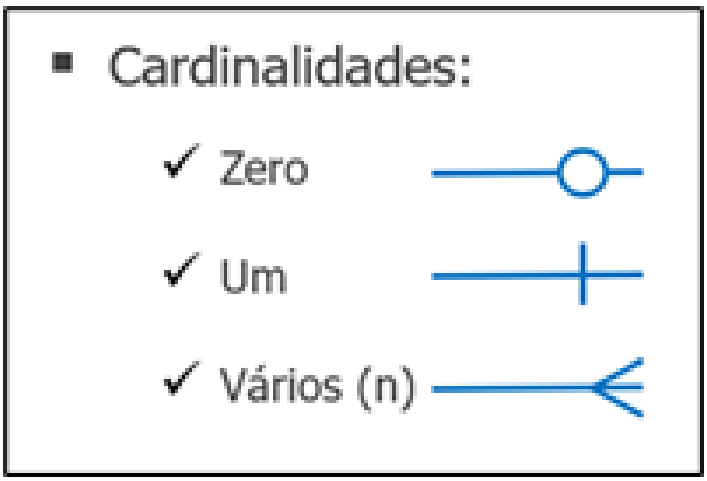


Figura 3.18: Notação da engenharia da informação para a representação das cardinalidades dos relacionamentos.

Como pode ser observado na figura 3.18, diferentemente da notação proposta por Peter Chen, na engenharia da informação, a representação das cardinalidades dos relacionamentos é *gráfica*, não é numérica. A cardinalidade *zero* (0) é representada por um círculo, a cardinalidade *um* (1) é representada por uma pequena linha vertical e a cardinalidade *n* é representada por três pequenas linhas, uma horizontal mais duas inclinadas, que se encontram em um ponto.

Para facilitar o entendimento da notação da engenharia da informação, as figuras 3.19 e 3.20 apresentam o mesmo modelo representado pelas notações de Peter Chen e da engenharia da informação.

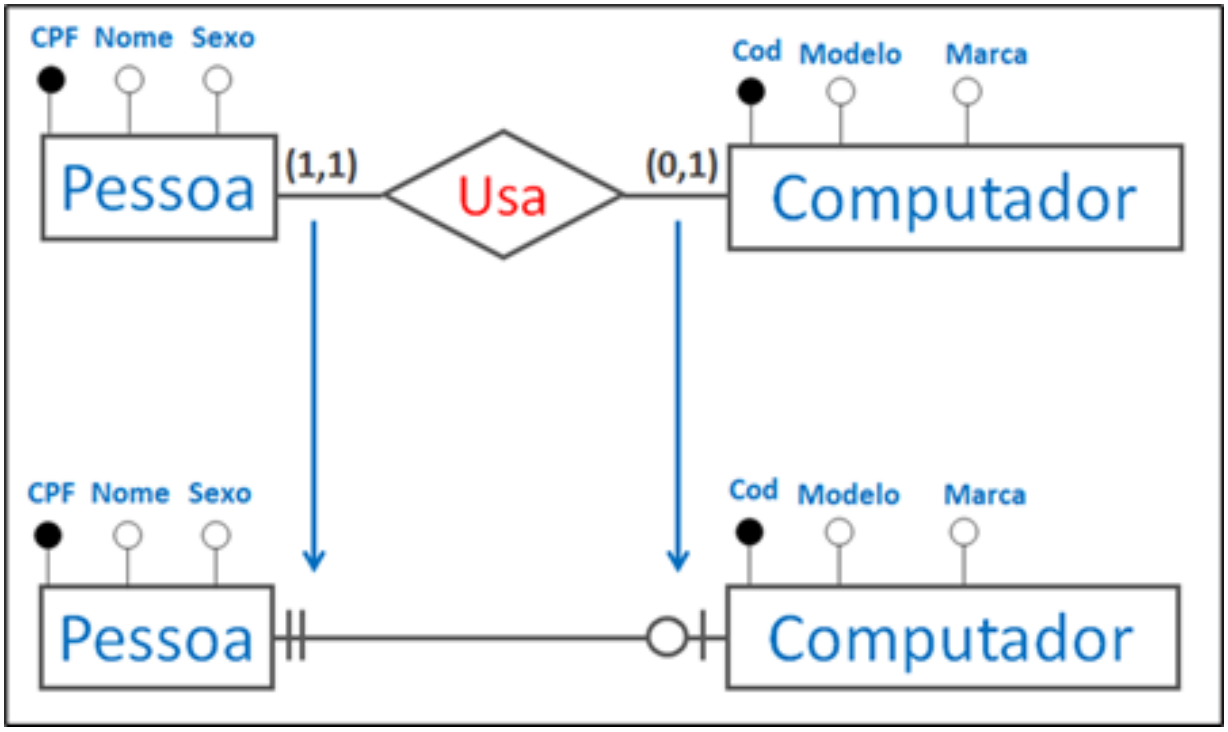


Figura 3.19: Representação das cardinalidades de relacionamentos 0 e 1 segundo a notação de Peter Chen e a da Engenharia da informação.

Como pode ser observado, a parte superior da figura 3.19 apresenta o modelo conceitual com o relacionamento `Usa`, de cardinalidade 1:1, entre as entidades `Pessoa` e `Computador` de acordo com notação proposta por Peter Chen. A parte inferior da figura apresenta o mesmo modelo de acordo com a notação da engenharia da informação.

Pode-se reparar que a principal diferença entre os dois modelos é que as cardinalidades mínimas e máximas são representadas pelos números 0 e 1, no DER segundo Peter Chen, e por um círculo (para representar o número 0) e uma pequena linha vertical (para representar o número 1), na notação da engenharia da informação. Outra diferença existente entre os dois modelos consiste na ausência do losango para representar o relacionamento `Usa` na notação da engenharia da informação.

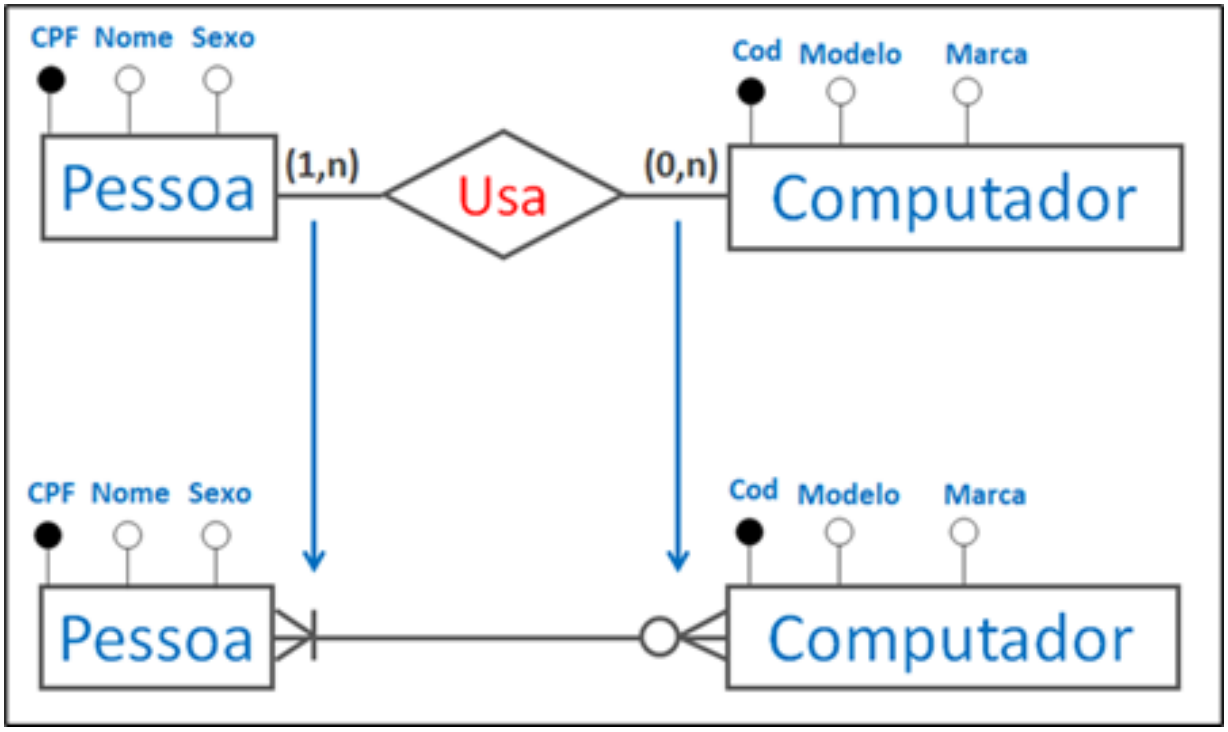


Figura 3.20: Representação das cardinalidades de relacionamentos 0, 1 e n, segundo as notações de Peter Chen e da engenharia da informação.

A figura 3.20 traz o mesmo modelo apresentado na figura 3.19, porém as cardinalidades máximas do relacionamento foram alteradas para *n:n*. Pode-se observar que a principal diferença entre os dois modelos, tal como apresentado na figura 3.19, são as cardinalidades mínimas e máximas, que são representadas: pelos números 0 e 1 e pela letra *n*, no DER segundo Peter Chen; e por um círculo (para representar o número 0), uma pequena linha vertical (para representar o número 1) e três linhas com um ponto de interseção (para representar a letra *n*), na notação da engenharia da informação.

O relacionamento de herança também possui algumas diferenças de representação segundo a notação de Peter Chen e a da engenharia da informação. A figura 3.21 apresenta o relacionamento de herança.

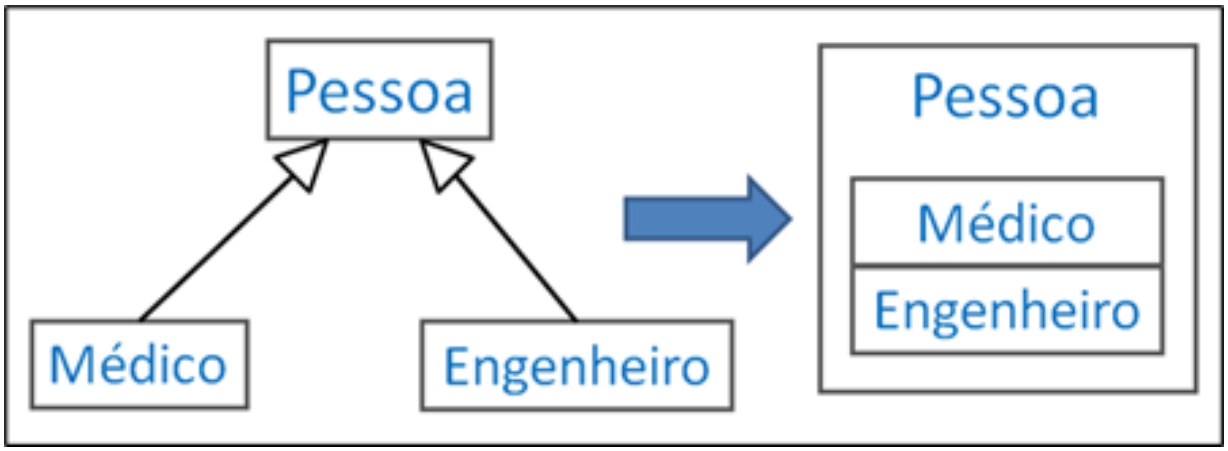


Figura 3.21: Representação do relacionamento de herança segundo as notações de Peter Chen e da engenharia da informação.

Como pode ser observado, o lado esquerdo da figura 3.21 apresenta o relacionamento de herança de acordo com notação proposta por Peter Chen. Nessa notação, a entidade genérica `Pessoa` está conectada às entidades especializadas `Médico` e `Engenheiro` por meio de uma linha com um triângulo na extremidade. Por outro lado, na notação da engenharia da informação, as entidades especializadas são contidas na entidade genérica.

3.3 Exemplo de projeto conceitual

O projeto conceitual tem por objetivo gerar um modelo de dados que represente os dados gerados ou consumidos pelos processos de negócio do ambiente observado. Esse projeto inicia-se com o levantamento dos requisitos que farão parte do escopo do projeto.

A seguir, será apresentado um enunciado bem simples que simula os requisitos a serem contemplados por um modelo conceitual.

Enunciado: Deseja-se desenvolver um sistema de informação computadorizado para uma companhia aérea com os seguintes requisitos:

- Toda pessoa cadastrada no sistema deve possuir CPF e nome; quando possível, deve-se cadastrar o sexo também, embora existam vários outros sexos, para simplificar o modelo, serão usados apenas os valores ‘M’, para masculino, e ‘F’, para feminino;
- Os pilotos devem cadastrar sua licença de voo e os passageiros, a sua categoria;
- Deve-se cadastrar também os telefones dos passageiros;
- Deve-se cadastrar os dados de voo, número e data, e o respectivo piloto;
- Deve-se identificar também os passageiros dos voos.

Solução

- ***Toda pessoa cadastrada no sistema deve possuir CPF e nome e; quando possível, deve-se cadastrar o sexo também, cujos valores podem ser ‘M’, para masculino, e ‘F’, para feminino.***

A partir da descrição dos requisitos apresentada anteriormente, podemos criar a entidade `Pessoa` com os atributos "CPF", "Nome" e "Sexo", conforme apresentado na figura 3.22. Essa entidade possui os atributos obrigatórios "CPF" e "Nome" e o atributo opcional "Sexo". Além de ser opcional, o atributo "Sexo" pode receber apenas um conjunto restrito de valores. Essas características do atributo "Sexo" ficarão bem evidenciadas futuramente, quando implementarmos o projeto físico.

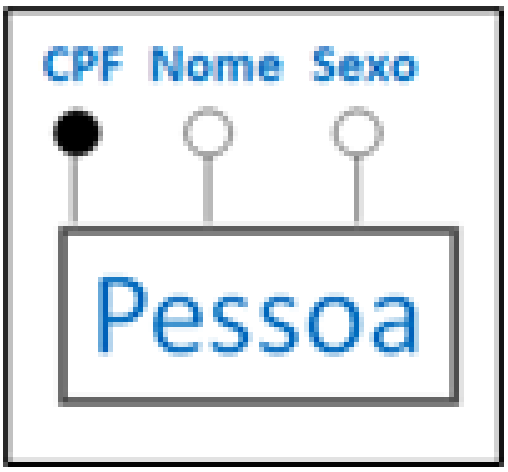


Figura 3.22: Entidade "Pessoa".

- ***Os pilotos devem cadastrar sua licença de voo e os passageiros, a sua categoria.***

A descrição apresentada cita apenas as entidades `Piloto` e `Passageiro`, porém podemos inferir que pilotos e passageiros também são *pessoas*, mas possuem alguns atributos específicos. Essa característica nos leva a criar o relacionamento de herança, conforme apresentado na figura 3.23.

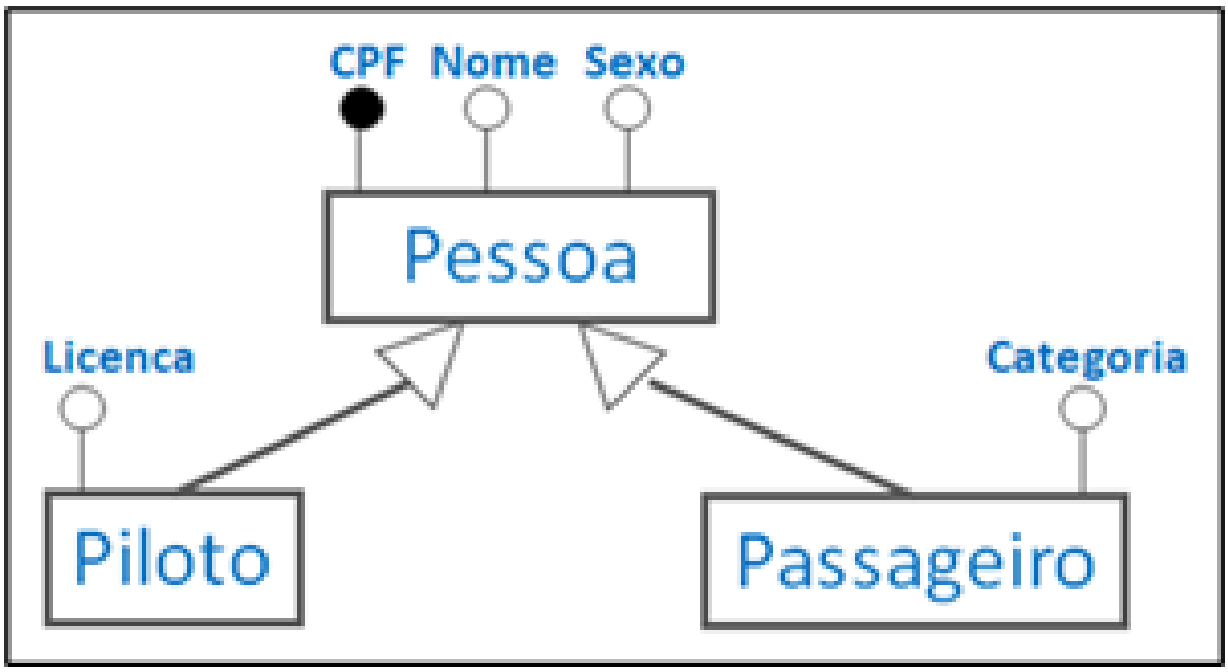


Figura 3.23: Relacionamento de herança.

- ***Deve-se cadastrar também os telefones dos passageiros.***

Esse requisito nos leva a uma nova entidade, a entidade `Telefone`. Nesse caso, consideramos que um passageiro pode possuir vários telefones, mas um telefone pode pertencer a somente um passageiro, o que nos levou a criar um relacionamento de cardinalidade *1:n* entre as entidades `Passageiro` e `Telefone`, conforme apresentado na figura 3.24.

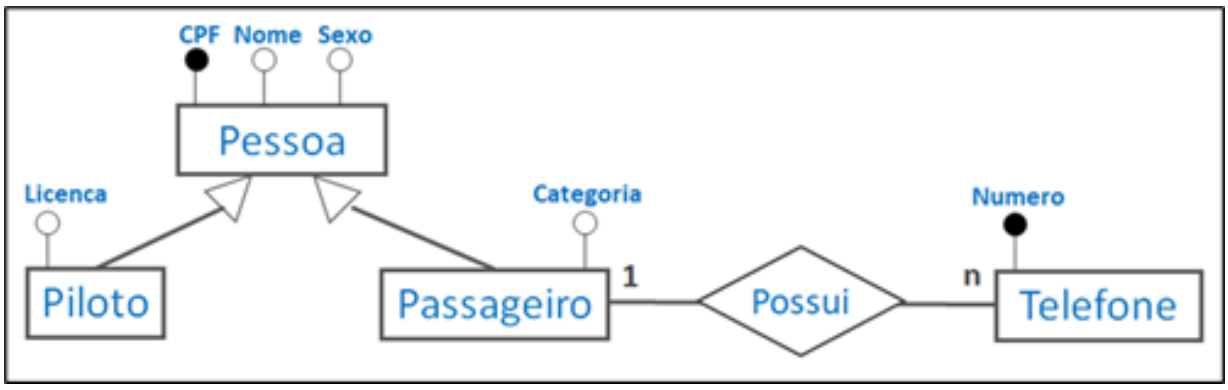


Figura 3.24: Relacionamento de cardinalidade 1:n entre as entidades "Passageiro" e "Telefone".

- ***Deve-se cadastrar os dados de voo, número e data, e o respectivo piloto.***

Esse requisito sugere a criação de uma nova entidade, `Voo`, e um relacionamento de cardinalidade *1:n* entre as entidades `Piloto` e `Voo`, uma vez que um voo é pilotado por apenas um piloto, porém um piloto pode pilotar vários voos. Repare que a cardinalidade mínima de `Voo` é 1, pois todo voo precisa obrigatoriamente possuir um piloto. A figura 3.25 apresenta essa parte do modelo.

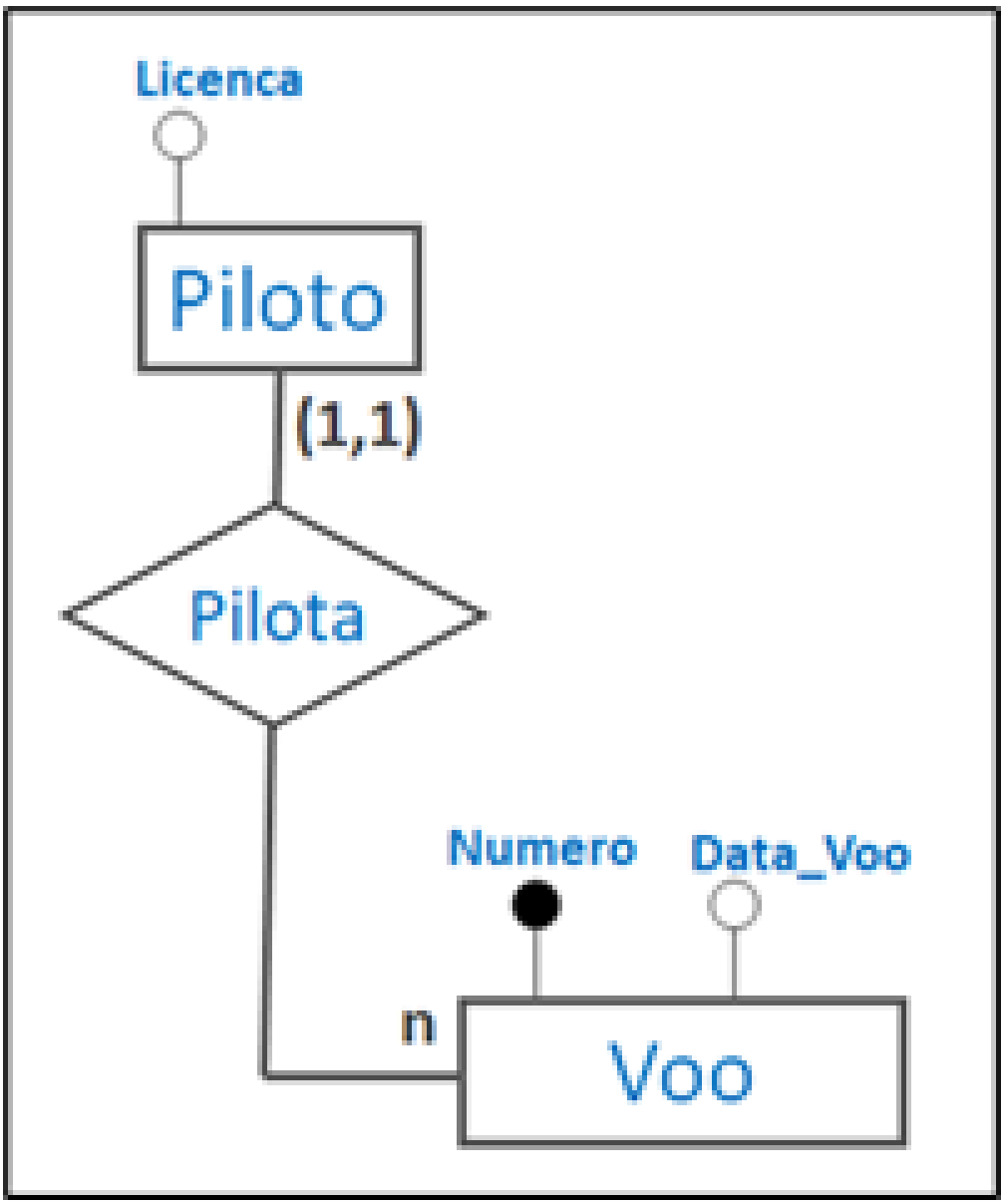


Figura 3.25: Entidade "Voo" e um relacionamento de cardinalidade 1:n entre as entidades "Piloto" e "Voo".

- ***Deve-se identificar também passageiros dos voos.***

Esse requisito sugere a criação de um relacionamento denominado `Reserva`, de cardinalidade *n:n* entre as entidades `Passageiro` e `Voo`, pois um voo pode ser reservado por vários passageiros e um passageiro pode reservar vários voos. A figura 3.26 apresenta essa parte do modelo. Cabe ressaltar que as decisões sobre nome e a cardinalidade de relacionamentos pode e deve ser discutida com os

usuários do sistema, pois são eles que conhecem o ambiente observado.

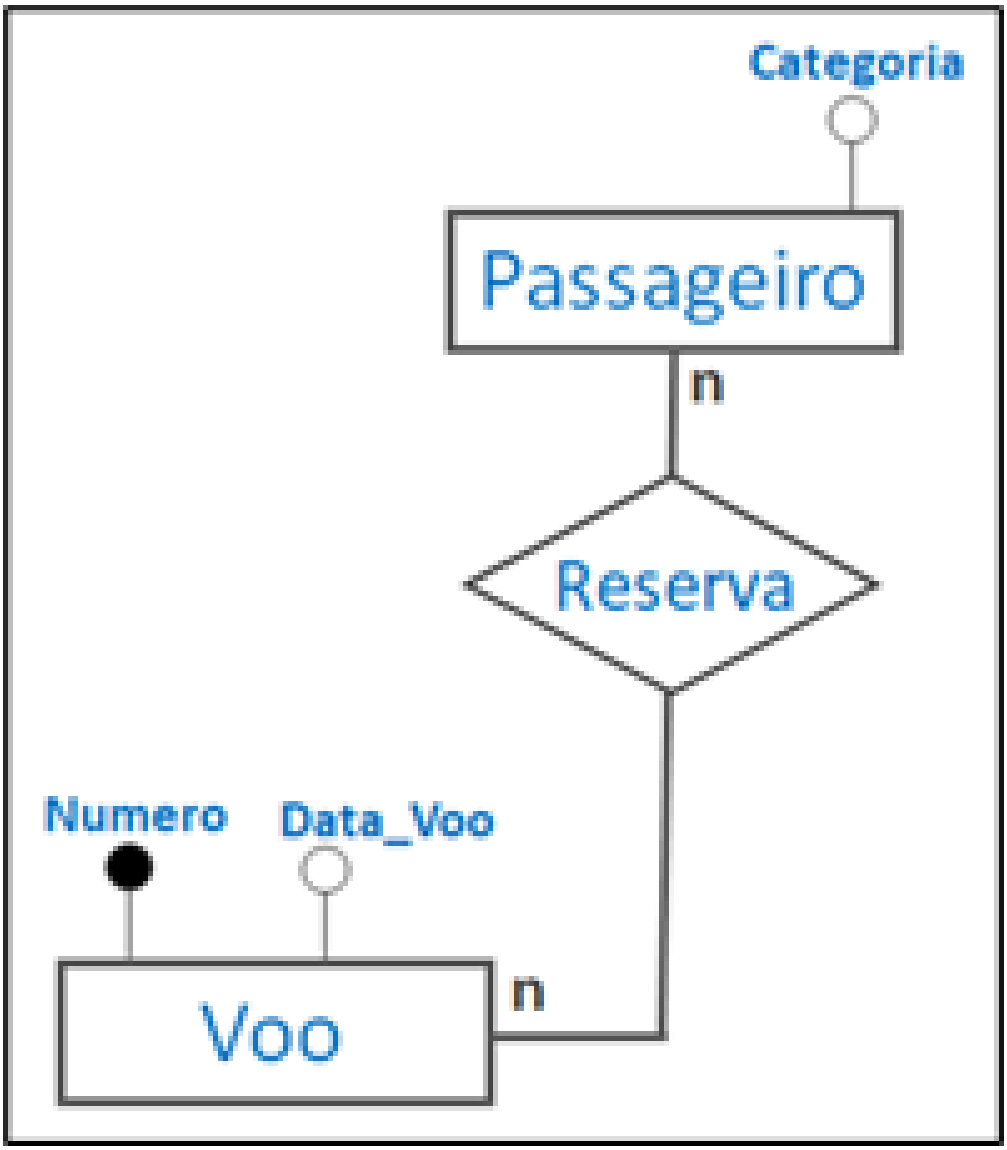


Figura 3.26: Relacionamento de cardinalidade n:n entre as entidades "Passageiro" e "Voo".

Finalmente, a figura 3.27 apresenta o modelo completo que contempla os requisitos descritos no enunciado.

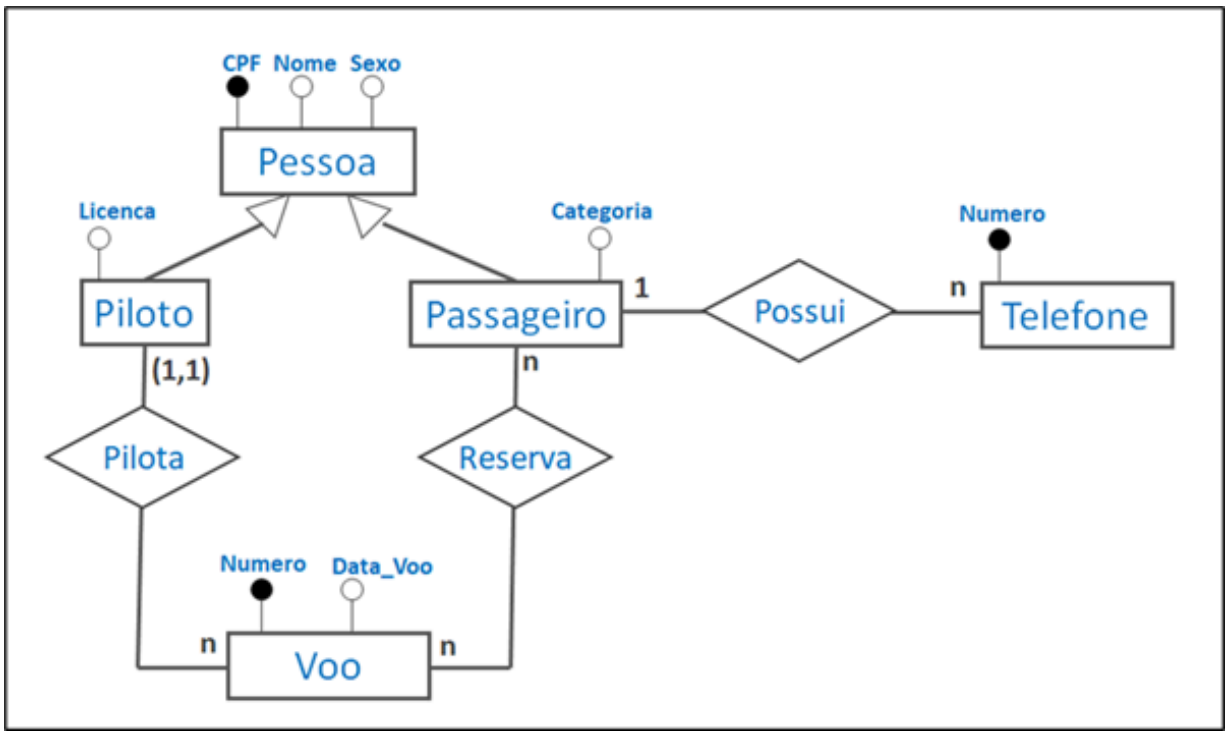


Figura 3.27: Modelo conceitual completo.

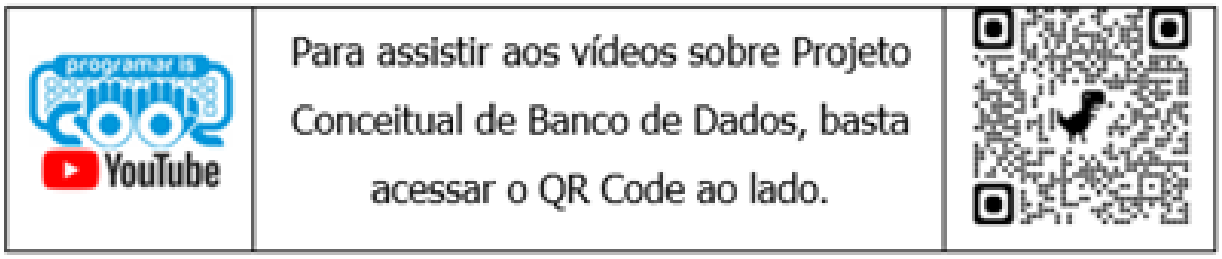


Figura 3.28: Vídeos sobre Projeto Conceitual de Banco de Dados. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cICCnNIiFan8Hvgi9YIUZlf1

Referências

CHEN, P. *Modelagem de Dados Modelagem De Dados* - A Abordagem Entidade - Relacionamento Para Projeto Lógico. São Paulo: Makron Books, 1990.

HEUSER, C. A. *Projeto de banco de dados*. Porto

Alegre: Bookman, 2009.

MARTIN, J. *Engenharia da Informação*: Introdução. Rio de Janeiro: Campus, 1991.

CAPÍTULO 4
Projeto lógico de banco de dados

O objetivo do projeto lógico é construir o modelo lógico de dados. Esse modelo consiste no mapeamento do modelo conceitual para um paradigma específico, que neste caso é o relacional, mas poderia ser outro, como o paradigma orientado a objetos. Assim, o modelo lógico, diferentemente do modelo conceitual é dependente de paradigma, mas também é independente de SGBD específico.

4.1 Modelo Relacional

O paradigma relacional representa os dados em um banco de dados como uma coleção de relações (tabelas). Nesse modelo, cada linha é denominada *tupla*; uma coluna é chamada de *atributo*; a tabela é chamada de *relação*.

Uma tabela consiste em um conjunto desordenado de tuplas exclusivas. Tabelas são compostas por linhas, colunas e chaves primárias. As chaves primárias garantem a unicidade de cada tupla. Os relacionamentos são implementados por meio de chaves estrangeiras. As linhas de uma tabela não estão ordenadas, e os valores dos campos devem ser *atômicos*, ou seja, *simples* (não são divisíveis) e monovalorados (só possuem um valor). A figura 4.1 apresenta a tabela `Pessoa` e seu relacionamento com a tabela `Computador`.

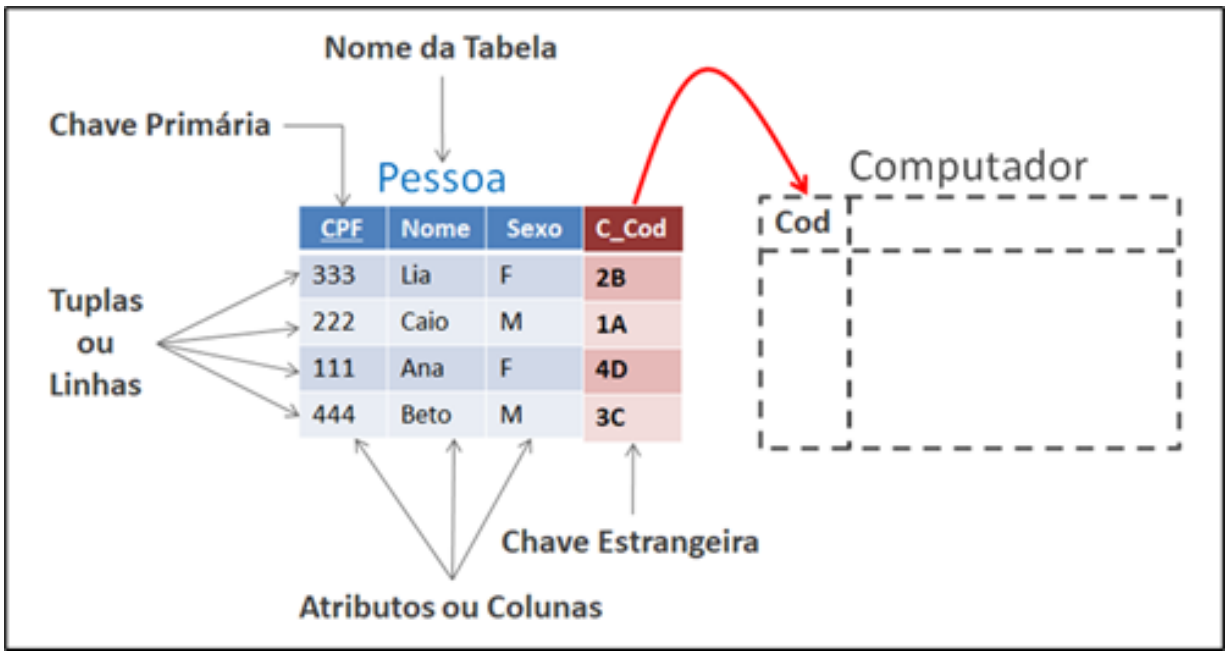


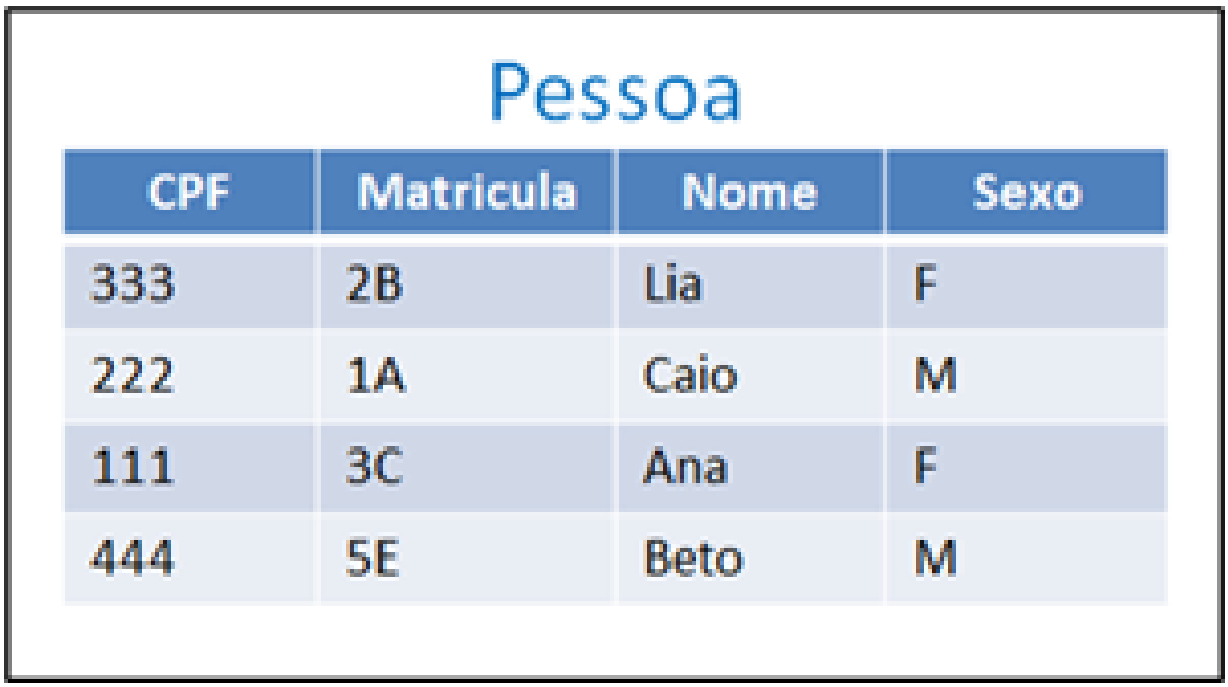
Figura 4.1: Tabela "Pessoa" e seu relacionamento com a tabela "Computador".

Chaves

No modelo relacional, são considerados os seguintes tipos de chaves:

- Candidatas;
- Primárias;
- Alternativas ou alternadas;
- Estrangeiras.

Chave candidata é uma coluna ou combinação de colunas (pois podem ser chaves compostas) cujos valores distinguem uma linha das demais dentro de uma tabela, porém ainda não foram escolhidas para serem chaves primárias. A figura 4.2 apresenta um exemplo de tabela com suas respectivas chaves candidatas.



Pessoa

CPF	Matricula	Nome	Sexo
333	2B	Lia	F
222	1A	Caio	M
111	3C	Ana	F
444	5E	Beto	M

Figura 4.2: Tabela "Pessoa" com suas chaves candidatas.

Como pode ser observado na figura 4.2, as colunas *CPF* e *Matrícula* da tabela `Pessoa`, separadamente, podem ser utilizadas como campos únicos capazes de diferenciar uma tupla (linha) das demais. Então, cada uma dessas colunas é denominada chave candidata e pode ser escolhida para ser chave primária dessa tabela.

Ao determinarmos que *CPF* será a chave primária da tabela `Pessoa`, automaticamente a coluna `Matrícula` passa a ser denominada **chave alternada** ou **alternativa**; ou seja, a coluna `Matrícula` era uma chave candidata que não foi escolhida para ser chave primária, mas possui todas as características necessárias.

Cabe ressaltar que as chaves candidatas, primárias e alternadas podem ser compostas, pois há situações em que um campo não é suficiente para diferenciar uma tupla das demais. No entanto, uma chave primária composta deve ser irredutível, ou seja, caso um campo que compõe uma chave primária composta seja retirado, ela deixa de ser chave primária por não ser mais capaz de diferenciar

uma tupla das demais. A figura 4.3 apresenta uma tabela com chave primária composta.

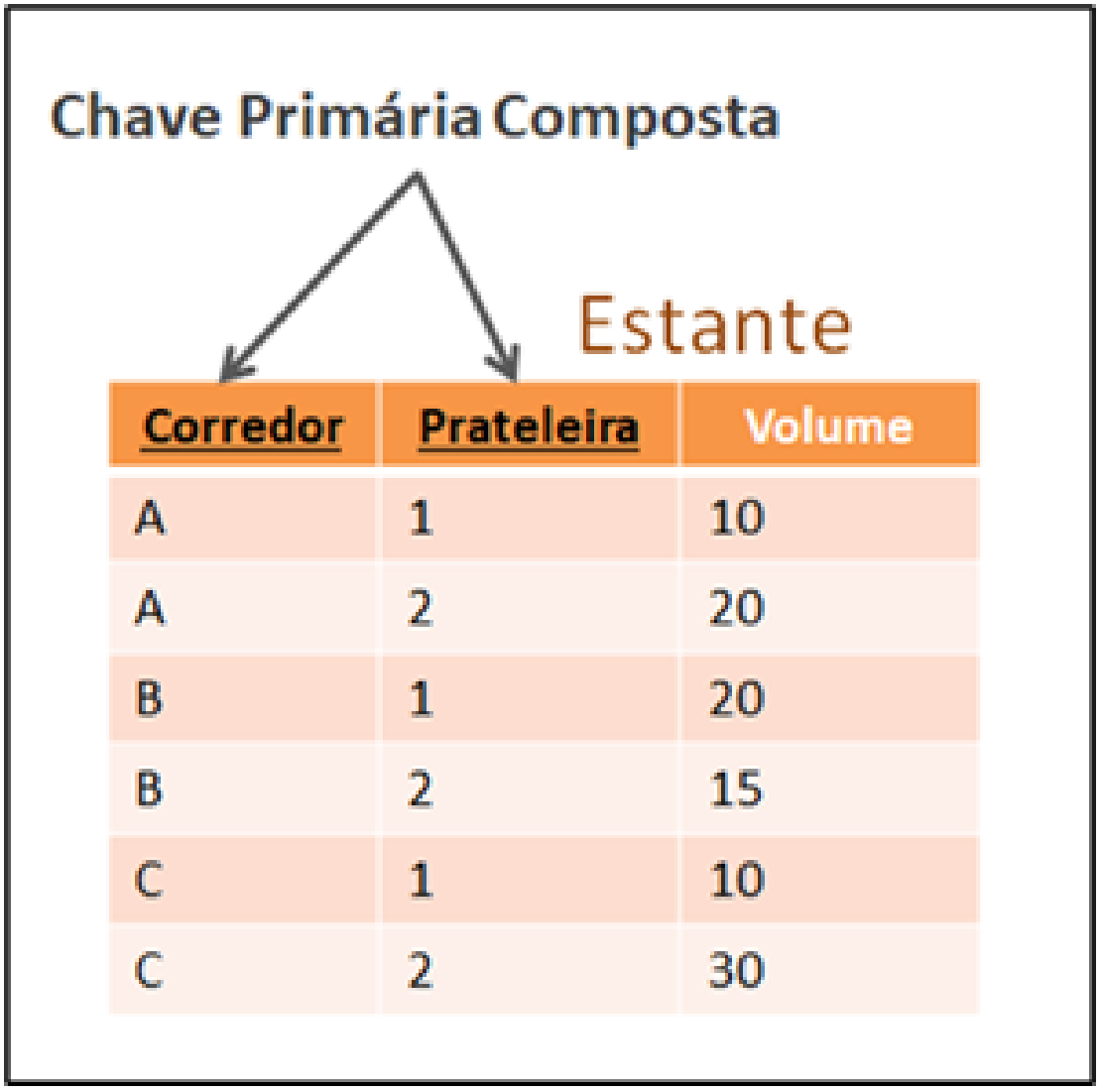


Figura 4.3: Tabela "Estante" com suas chaves primária composta.

Como pode ser observado na figura 4.3, a tabela `Estante` possui uma chave primária composta pelas colunas *Corredor* e *Prateleira*. Reparem que apenas o corredor não é capaz de identificar univocamente o volume armazenado na estante, pois é necessário saber em qual prateleira esse volume se encontra. Por outro lado, vários corredores possuem a mesma denominação de prateleira — neste caso, os números 1 e 2. Então, para localizar um volume, será

necessário saber em que prateleira de que corredor esse volume se encontra, caracterizando a necessidade de uma chave primária composta.

Em resumo, as principais características de uma chave primária são:

- **Unicidade**: uma tabela não pode possuir duas linhas com o mesmo valor para as suas chaves;
- **Obrigatoriedade**: os campos de uma chave primária não podem ser nulos;
- **Ser irredutível**: uma chave primária composta deve possuir o número mínimo de campos que garantam a sua unicidade, nenhum campo a mais.

Existe também o conceito de **chave estrangeira**, que é o mecanismo que permite a implementação de relacionamentos em um banco de dados relacional. Chave estrangeira é a coluna ou combinação de colunas cujos valores aparecem na chave primária da tabela referenciada. A figura 4.4 apresenta um exemplo de chave estrangeira para representar um relacionamento de cardinalidade 1:1.

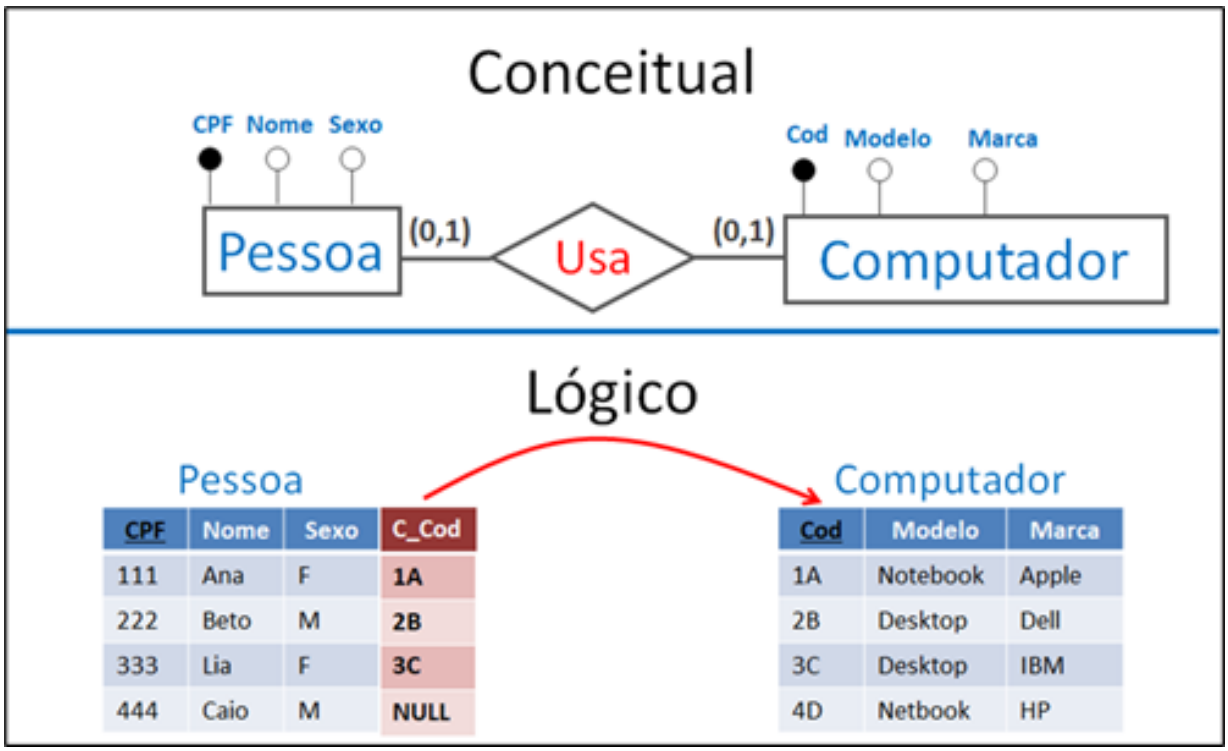


Figura 4.4: Chave estrangeira para representar um relacionamento de cardinalidade 1:1.

Como pode ser observado na figura 4.4, o relacionamento de cardinalidade 1:1 no modelo conceitual (parte superior da figura) foi mapeado para uma chave estrangeira no modelo lógico (parte inferior da figura). Assim, é possível identificar qual tupla da tabela `Pessoa` está relacionada a uma tupla da tabela `Computador`.

A chave estrangeira é o campo `C_Cod` da tabela `Pessoa`, que referencia o campo `Cod` na tabela `Computador`; ou seja, ao inserirmos um registro na tabela `Pessoa`, se alimentarmos o campo `C_Cod`, o valor inserido deverá existir no campo `Cod` da tabela `Computador`. Esse recurso é um tipo de restrição de integridade, denominado *restrição de integridade referencial*, e é garantido pelo SGBD.

Restrições de integridade

Restrição de integridade é uma regra de consistência de dados que é garantida pelo próprio SGBD. As restrições de integridade básicas são:

- **Integridade de Domínio:** define os valores que podem ser assumidos pelos campos de uma coluna;
- **Integridade de Vazio:** especifica se os campos de uma coluna podem ou não ser vazios;
- **Integridade de Unicidade:** define que o valor do campo ou campos são únicos;
- **Integridade de Chave:** define que os valores da chave primária e alternativa devem ser únicos;
- **Integridade de Entidade:** define que nenhum valor da chave primária pode ser nulo (campo obrigatório);
- **Integridade Referencial:** define que os valores dos campos que aparecem numa chave estrangeira devem aparecer na chave primária da tabela referenciada. Esse conceito está representado na figura 4.4.

Essas restrições são garantidas automaticamente pelo SGBD Relacional, o programador não precisa implementá-las.

A figura 4.5 apresenta os conceitos de integridade de domínio, de vazio e de unicidade:

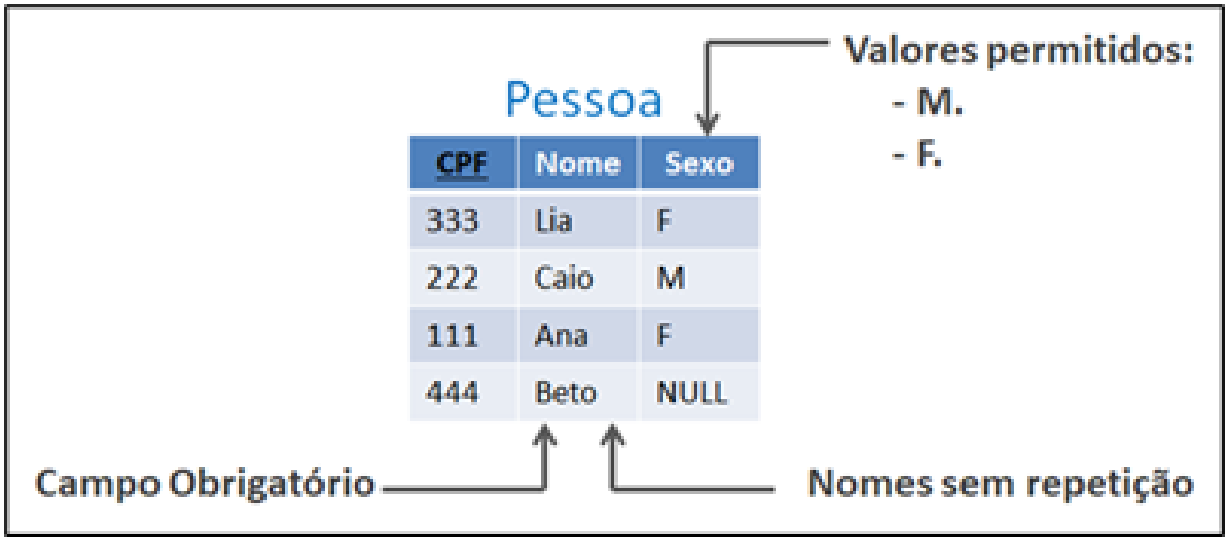


Figura 4.5: Integridade de domínio, de vazio e de unicidade.

Conforme apresentado na figura 4.5, o atributo *Nome* é obrigatório e único, ou seja, toda tupla deve possuir um conteúdo para o atributo

Nome e não pode haver dois registros com o mesmo valor para esse atributo, caracterizando as restrições de vazio e unicidade, respectivamente. Por outro lado, o atributo *Sexo* pode possuir apenas os valores 'M' ou 'F', caracterizando a restrição de domínio, ou seja, esse campo só poderá receber valores específicos.

A figura 4.6 apresenta os conceitos de integridade de chave e de entidade:

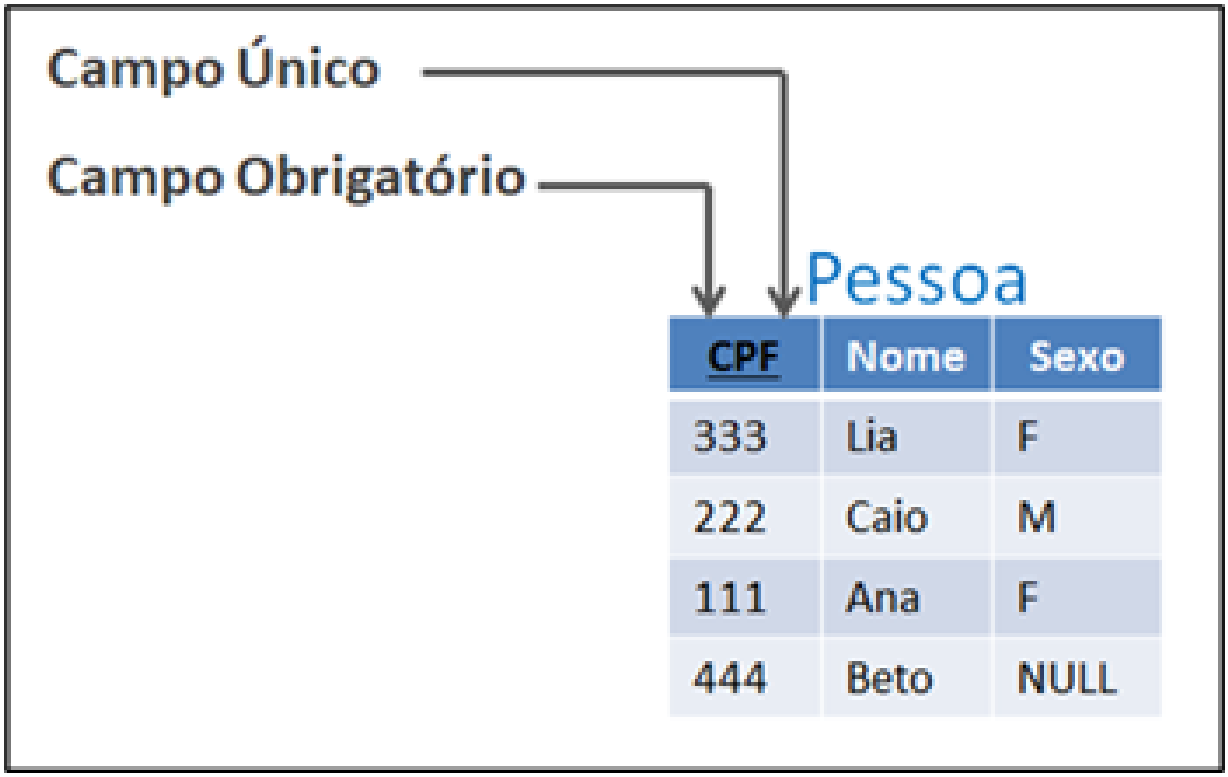


Figura 4.6: Integridade de chave e de entidade.

A figura 4.6 apresenta as restrições de chave e de entidade, ambas são peculiares à chave primária. A restrição de chave garante que a chave primária seja única, isto é, que não haja duas chaves primárias idênticas na tabela `Pessoa`. A restrição de entidade assegura que todos os campos que compõem a chave primária possuam conteúdo. Quando abordarmos o modelo físico, mostraremos como essas restrições são implementadas no SGBD.

4.2 Mapeamento do modelo conceitual para o lógico

Como já foi citado anteriormente, o principal artefato do projeto lógico relacional é um modelo que represente o mapeamento do modelo conceitual para o nível lógico. A principal tarefa para a criação desse modelo, na maior parte das situações, é mapear as entidades para tabelas e os relacionamentos para chaves estrangeiras.

A seguir, serão apresentadas as estratégias do mapeamento do modelo conceitual para o modelo lógico relacional, organizadas pelas cardinalidades dos relacionamentos.

Cardinalidade 1:1

Os relacionamentos com cardinalidade 1:1 podem gerar três situações específicas: ser opcional para as duas entidades que participam do relacionamento; ser obrigatório para pelo menos uma das entidades; ou ser obrigatório para ambas. Cabe ressaltar que o relacionamento é opcional quando a cardinalidade mínima é 0 (zero) e obrigatório quando a cardinalidade mínima é 1 (um).

A figura 4.7 apresenta o mapeamento do modelo conceitual (parte superior) para o lógico (parte inferior) do relacionamento `Usa` com cardinalidade 1:1 entre as entidades `Pessoa` e `Computador`. Esse relacionamento é opcional para as duas entidades, ou seja, um computador pode estar associado a uma ou nenhuma pessoa e uma pessoa pode estar associada a um ou nenhum computador.

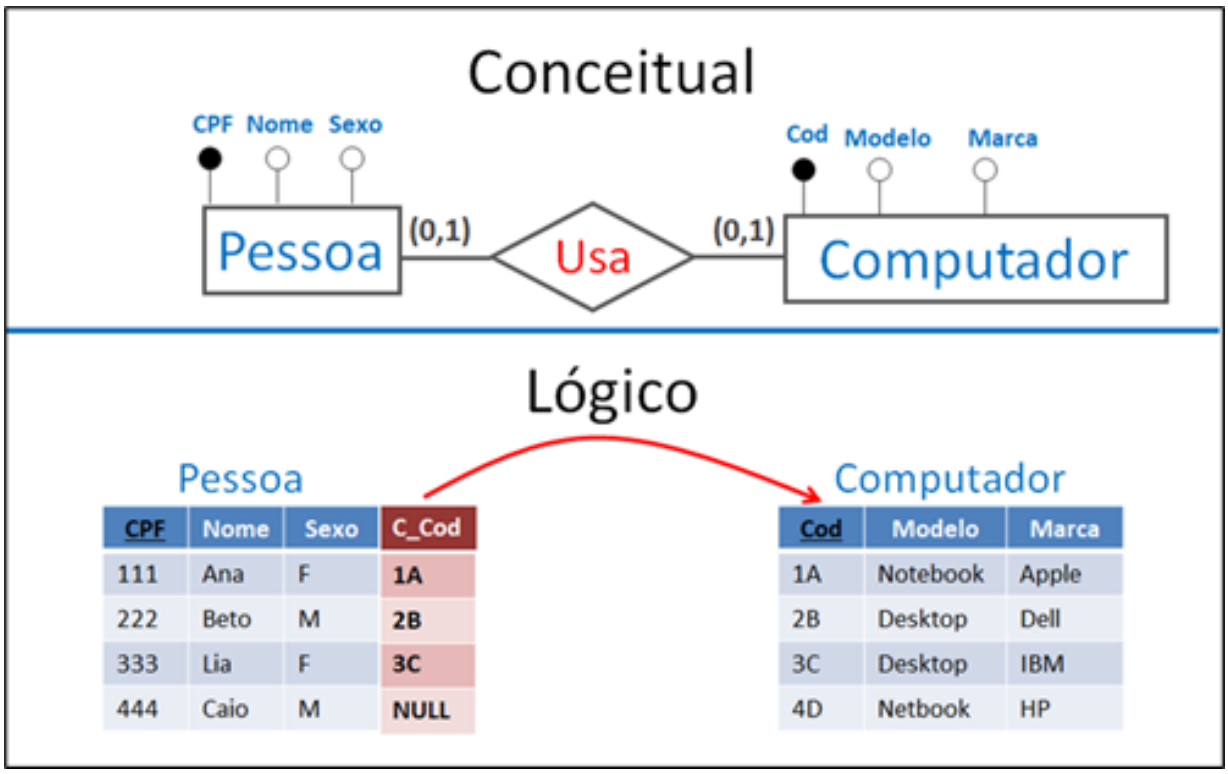


Figura 4.7: Mapeamento do modelo conceitual para o lógico de relacionamento com cardinalidade 1:1 opcional para as duas entidades.

Na figura 4.7, a entidade `Pessoa` no nível conceitual foi mapeada para a tabela `Pessoa` no nível lógico, e seus atributos foram mapeados para as colunas dessa tabela: seu atributo identificador (*CPF*) se tornou chave primária na tabela e o relacionamento `Usa` se transformou na coluna `C_Cod`, que é uma chave estrangeira na tabela `Pessoa`. Nessa figura, é possível observar que os valores existentes na coluna `C_Cod` da tabela `Pessoa` têm que existir na coluna `Cod` da tabela `Computador`, devido à restrição de integridade referencial. Essa restrição garante que, quando uma pessoa se relacionar com um computador, esse computador existirá na tabela `Computador`.

Outro aspecto importante a ser observado é que o valor da coluna `C_Cod`, que representa a chave estrangeira, pode ser vazio (*null*, que representa ausência de conteúdo), pois o modelo permite que haja pessoas sem estarem associadas a computadores, assim como permite que haja computadores sem estarem associados a pessoas,

pois a cardinalidade mínima é 0 (zero) em ambos os lados. É possível reparar que a pessoa cujo `CPF` é `444` não está associada a nenhum computador, e que o computador cujo `Cod` é `4D` não está associado a nenhuma pessoa.

Quando o relacionamento de cardinalidade 1:1 é opcional em ambos os lados, pode ser mapeado para uma chave estrangeira para qualquer uma das tabelas, porém em apenas uma delas. A figura 4.8 apresenta outra possibilidade de mapeamento do relacionamento apresentado na figura 4.7.

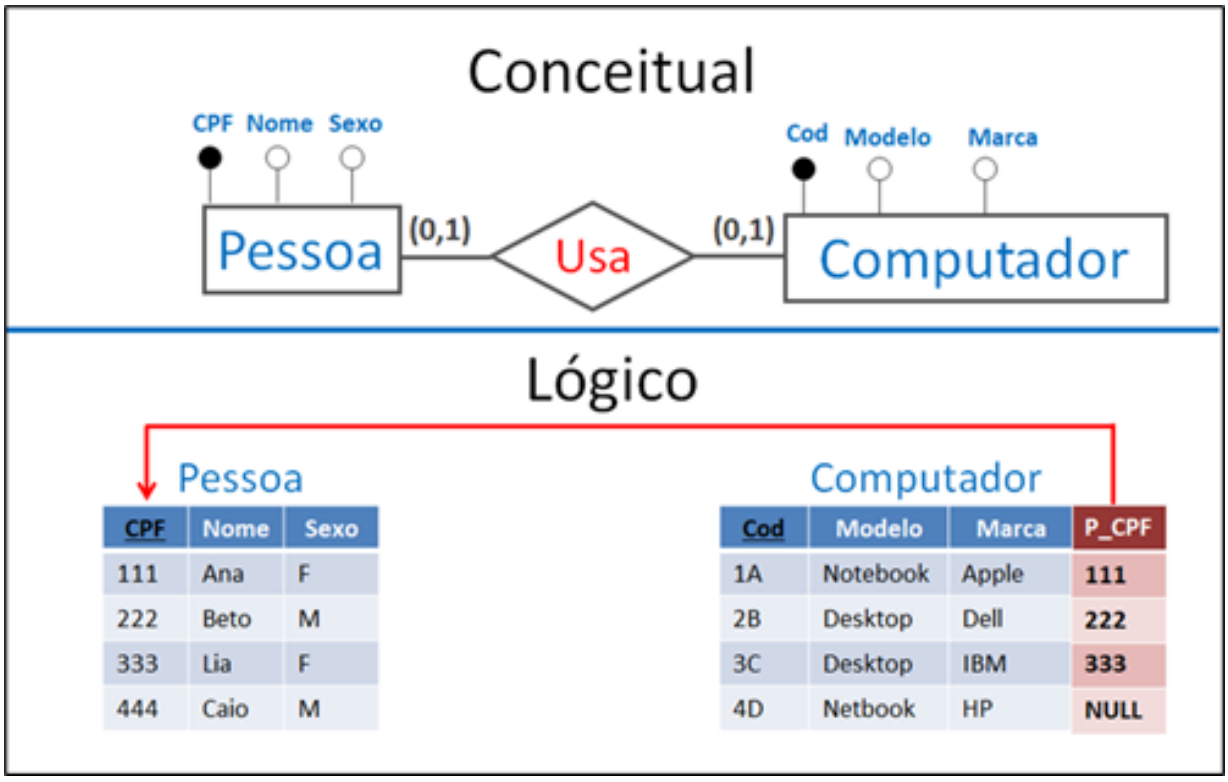


Figura 4.8: Mapeamento do modelo conceitual para o lógico de relacionamento com cardinalidade 1:1 opcional para as duas entidades.

A figura 4.8 apresenta o mesmo modelo conceitual apresentado na figura 4.7, porém o modelo lógico apresenta a chave estrangeira representada pela coluna `P_CPF` na tabela `Computador`, que referencia a coluna `CPF` na tabela `Pessoa`. Essa é outra possibilidade de mapeamento do relacionamento `Usa`, que é opcional em ambos os lados.

Quando uma das entidades participantes do relacionamento tem participação obrigatória, ou seja, tem cardinalidade mínima igual a um, existe um melhor lado para se criar a chave estrangeira ao mapearmos o relacionamento do modelo conceitual para o lógico. A figura 4.9 apresenta o relacionamento de cardinalidade 1:1 com participação obrigatória de uma das entidades.

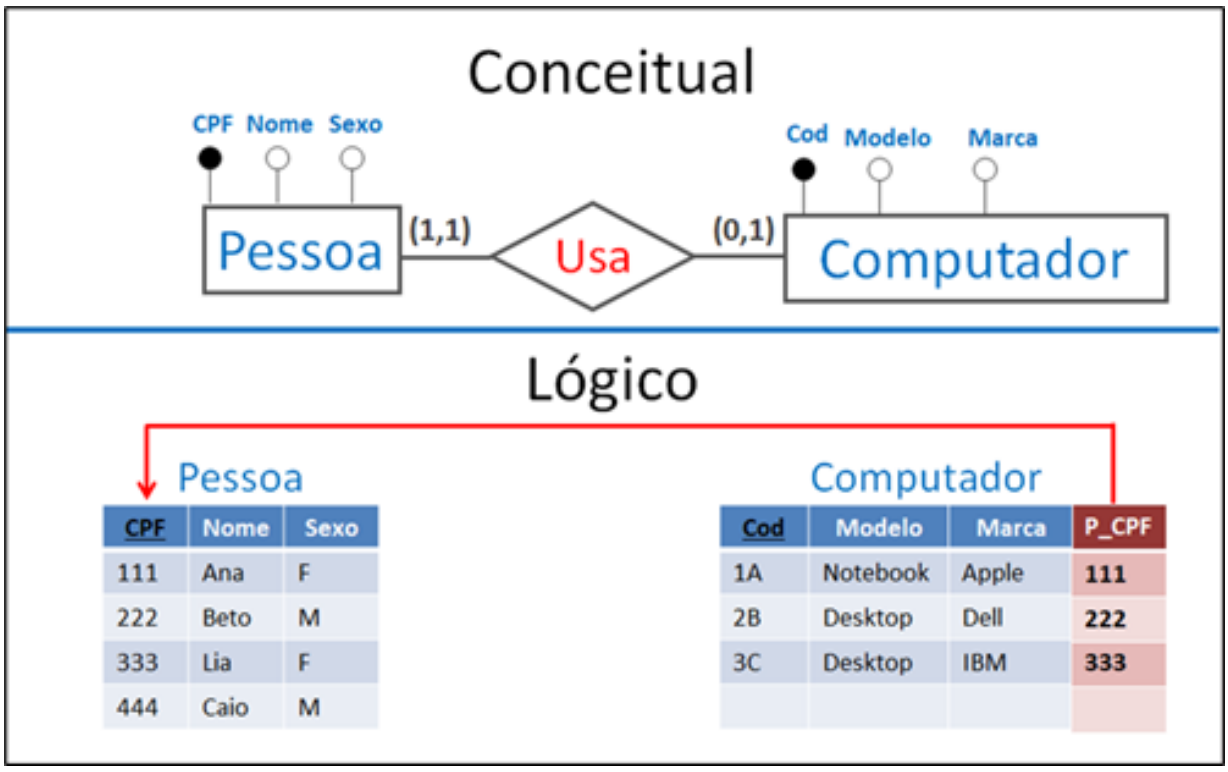


Figura 4.9: Relacionamento de cardinalidade 1:1 com participação obrigatória de uma das entidades.

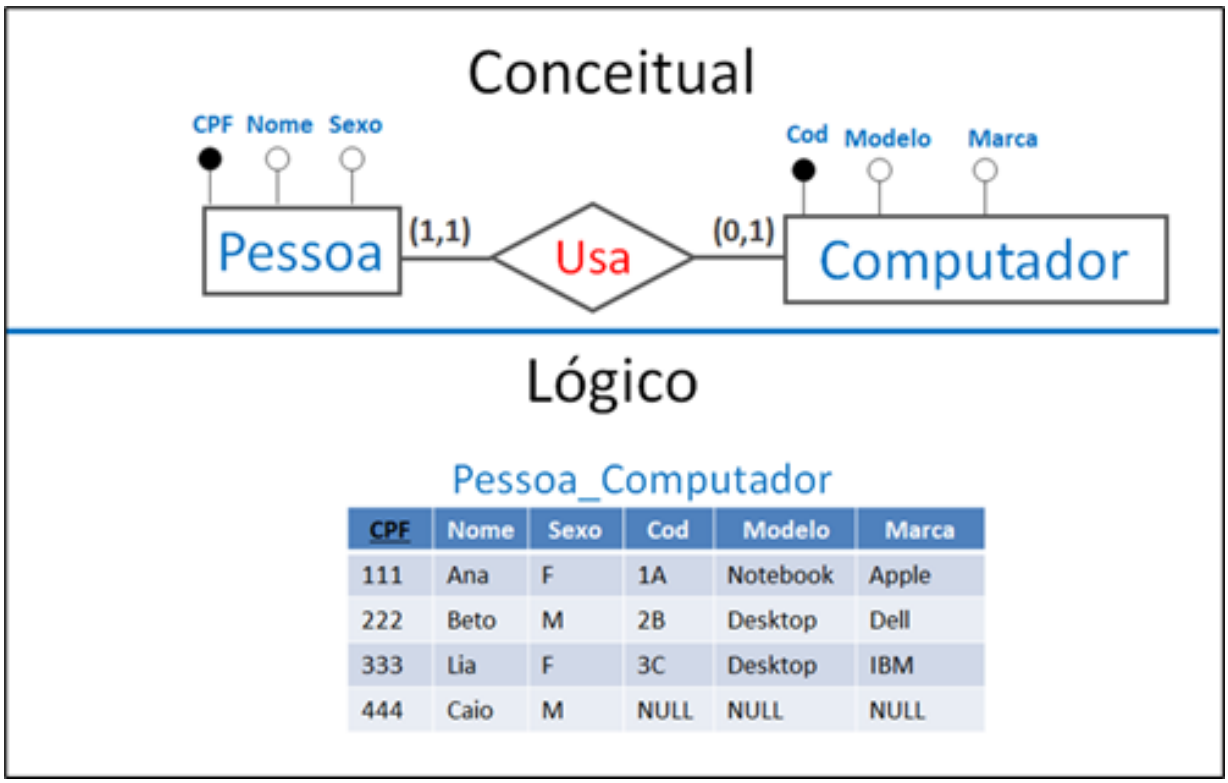
Na figura 4.9, a participação de `Pessoa` no relacionamento `Usa` é obrigatória devido ao fato de a cardinalidade mínima ser 1, ou seja, nesse modelo uma pessoa pode estar associada a um ou nenhum computador, porém um computador precisa estar associado a exatamente uma pessoa, por isso o relacionamento `Usa` foi mapeado para a coluna `P_CPF` da tabela `Computador`, que é chave estrangeira e é um campo obrigatório e único. A obrigatoriedade do campo garante que todos os computadores inseridos na tabela `Computador` tenham uma pessoa associada; já a unicidade restringe a apenas uma ocorrência de `Pessoa` em `Computador`, pois a

cardinalidade máxima é `1`. Caso isso não fosse feito, poderia haver uma pessoa com vários computadores.

Nessa figura, é possível reparar que a pessoa cujo CPF é "444" não está associada a nenhum computador, porém todos os computadores estão associados a exatamente uma pessoa devido à obrigatoriedade e unicidade da coluna `P_CPF` na tabela `Computador`.

Caso a participação obrigatória no relacionamento `Usa` fosse da entidade `Computador`, o relacionamento `Usa` seria mapeado para uma chave estrangeira na tabela `Pessoa`, que referenciaria `Cod` da tabela `Computador`.

Outra solução possível de mapeamento do modelo conceitual para o lógico quando o relacionamento possui cardinalidade 1:1 com participação obrigatória de pelo menos uma das entidades é a **fusão de tabelas**. A figura 4.10 apresenta essa estratégia de mapeamento.



CPF	Nome	Sexo	Cod	Modelo	Marca
111	Ana	F	1A	Notebook	Apple
222	Beto	M	2B	Desktop	Dell
333	Lia	F	3C	Desktop	IBM
444	Caio	M	NULL	NULL	NULL

Figura 4.10: Relacionamento de cardinalidade 1:1 com participação obrigatória de uma das entidades.

Na figura 4.10, assim como na figura 4.9, a participação de `Pessoa` no relacionamento `Usa` é obrigatória devido ao fato de a cardinalidade mínima ser 1. Nesse caso, as entidades `Pessoa` e `Computador` mais o relacionamento `Usa` foram mapeados para a tabela `Pessoa_Computador` — esse caso é denominado fusão de tabelas.

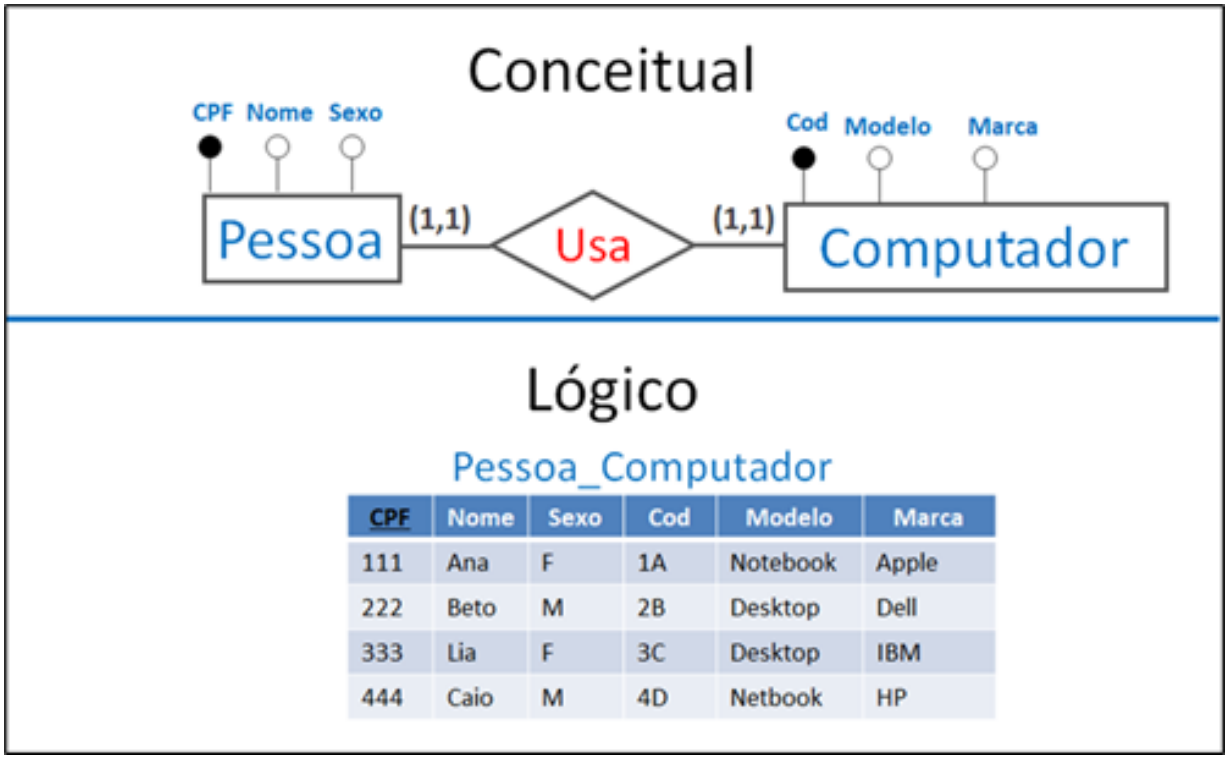
Nessa figura, é possível notar que a tabela `Pessoa_Computador` possui todos os atributos das entidades `Pessoa` e `Computador` e que a chave primária da tabela é o `CPF`, e isso se deve à participação obrigatória de `Pessoa` no relacionamento `Usa` com `Computador`.

Assim, todo registro da tabela `Pessoa_Computador` sempre terá os dados de `Pessoa`, mesmo que não tenha dados de computador, pois o CPF é obrigatório por ser chave primária da tabela. Como exemplo, existe o registro com o CPF "444", que possui os dados de `Pessoa`, mas não possui os dados de `Computador`, garantindo a regra da

obrigatoriedade de participação de `Pessoa`. Ou seja, pode existir uma pessoa sem estar associada a computador, mas não pode haver computador sem estar associado a uma pessoa.

Caso a participação obrigatória no relacionamento `Usa` fosse da entidade `Computador`, teríamos como chave primária da tabela `Pessoa_Computador` o atributo `Cod` da entidade `Computador`.

Quando as duas entidades participantes do relacionamento de cardinalidade 1:1 têm participação obrigatória, isto é, têm cardinalidades mínimas igual a 1, a melhor solução de mapeamento do modelo conceitual para o lógico é a fusão de tabelas, conforme mostrado na figura 4.11.



CPF	Nome	Sexo	Cod	Modelo	Marca
111	Ana	F	1A	Notebook	Apple
222	Beto	M	2B	Desktop	Dell
333	Lia	F	3C	Desktop	IBM
444	Caio	M	4D	Netbook	HP

Figura 4.11: Relacionamento de cardinalidade 1:1 com participação obrigatória das duas entidades.

Na figura 4.11, a chave primária da tabela `Pessoa_Computador` será `CPF` ou `Cod`, apenas um dos dois atributos, nunca será uma chave composta por esses dois campos, pois a chave primária deve ser

mínima, ou seja, deve possuir a quantidade mínima de campos. Nesse caso, todo registro da tabela possuirá dados de pessoa e de computador, pois ambos possuem participação obrigatória no relacionamento.

Na mesma figura, se o campo `CPF` for escolhido para ser chave primária, o campo `Cod` deverá ser único e obrigatório, pois ele será uma chave alternada. Assim, será garantido que uma pessoa se relacionará a exatamente um computador e que um computador se relacionará a exatamente uma pessoa.

Cardinalidade 1:n

Os relacionamentos com cardinalidade 1:n podem gerar várias situações. Abordaremos as duas mais comuns: a primeira, quando a entidade localizada do lado do relacionamento cuja cardinalidade é 1 possui participação opcional; e a segunda, quando a participação dessa entidade é obrigatória. A figura 4.12 apresenta a primeira situação.

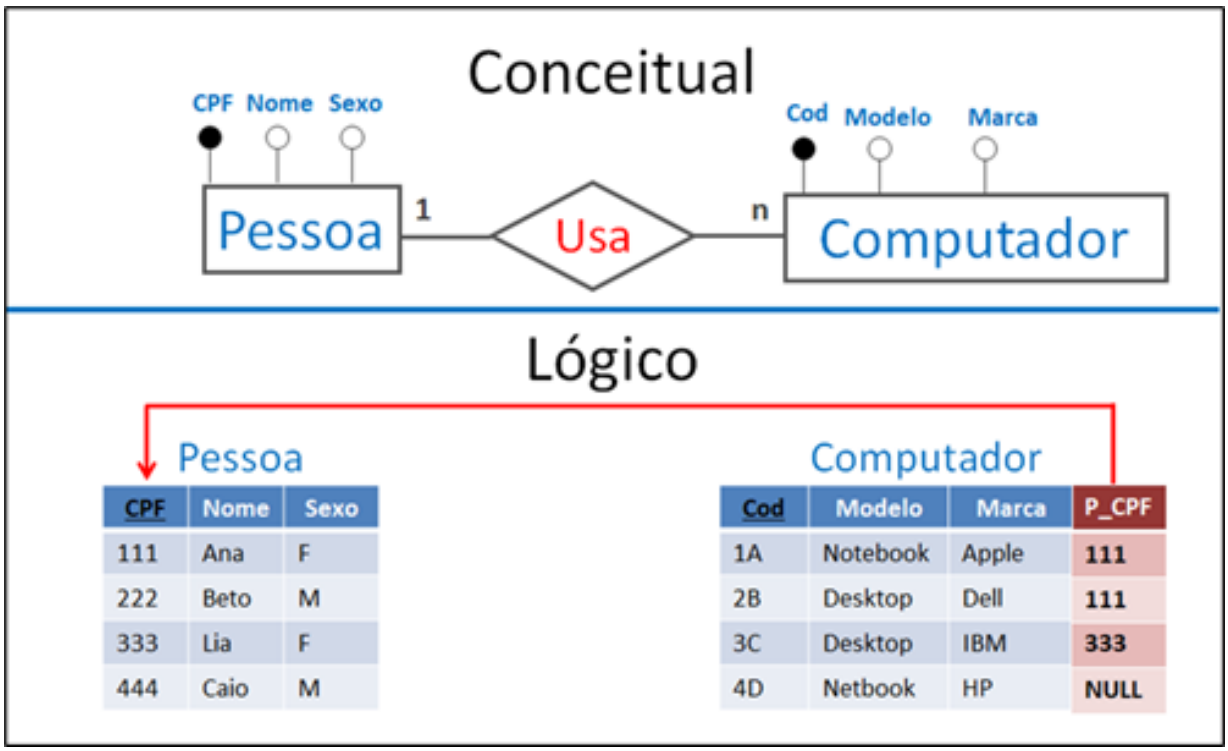


Figura 4.12: Relacionamento de cardinalidade 1:n com participação opcional das duas entidades.

Como pode ser observado na figura 4.12, uma pessoa pode estar relacionada a nenhum ou vários computadores e um computador pode estar relacionado a nenhuma ou uma pessoa. Nesse caso, ao mapear o modelo conceitual para o lógico, o relacionamento `Usa` de cardinalidade *1:n* foi mapeado para uma chave estrangeira na tabela que representa a entidade do lado de cardinalidade *n* do relacionamento. Assim, a coluna `P_CPF` é uma chave estrangeira que referencia a coluna `CPF` em `Pessoa`. A coluna `P_CPF` não é obrigatória, pois um computador pode não possuir uma pessoa relacionada, como é o caso do computador cujo `Cod` é `4D`, e pode possuir o mesmo `CPF` várias vezes, como é o caso do CPF "111", pois uma pessoa pode estar associada a vários computadores. Cabe ressaltar que uma pessoa não precisa estar associada a um computador, como é o caso do registro cujo `CPF` é "444".

A figura 4.13 apresenta a segunda situação, quando a entidade localizada do lado do relacionamento cuja cardinalidade é 1 possui

participação obrigatória.

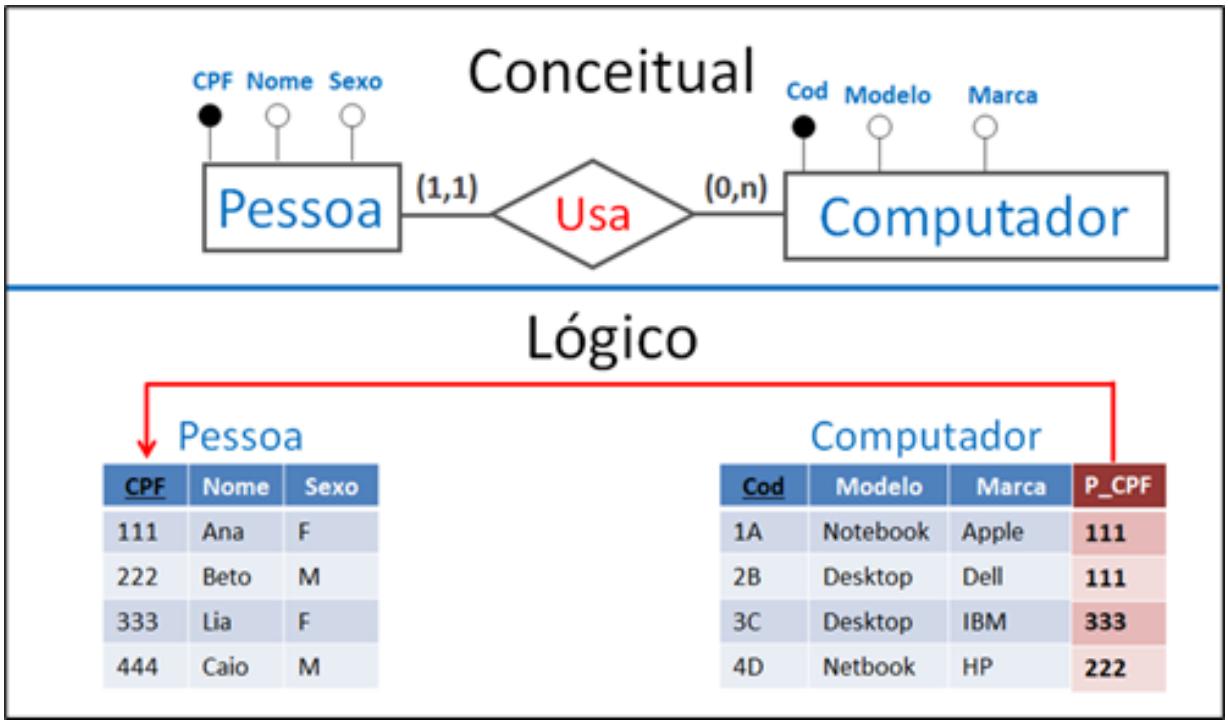
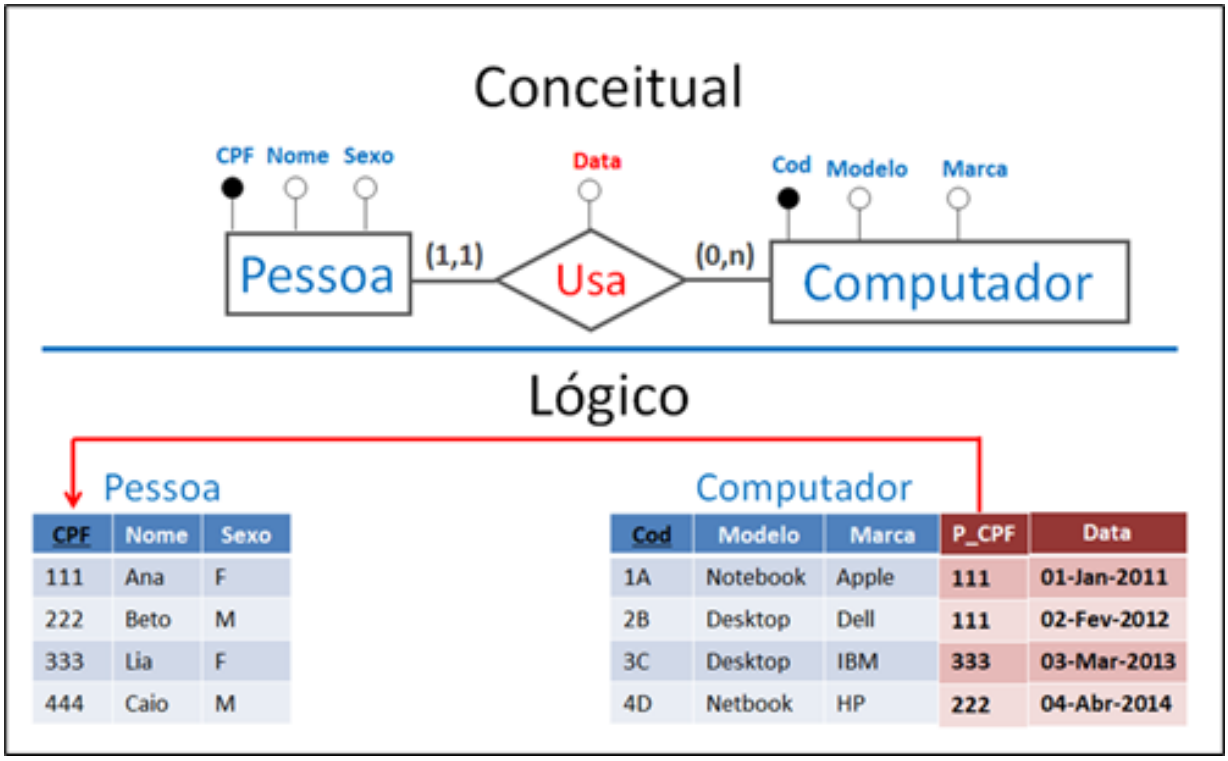


Figura 4.13: Relacionamento de cardinalidade 1:n com participação obrigatória de uma das entidades.

Como pode ser observado na figura 4.13, uma pessoa pode estar relacionada a nenhum ou vários computadores, e um computador deve estar relacionado a uma pessoa. Nesse caso, ao mapear o modelo conceitual para o lógico, o relacionamento `Usa` de cardinalidade *1:n* foi mapeado para uma chave estrangeira na tabela que representa a entidade do lado de cardinalidade *n* do relacionamento. Assim, a coluna `P_CPF` é uma chave estrangeira que referencia a coluna `CPF` em `Pessoa`. A coluna `P_CPF` é obrigatória, pois um computador deve possuir uma pessoa relacionada e pode possuir o mesmo `CPF` várias vezes (como é o caso do CPF "111"), pois uma pessoa pode estar associada a vários computadores. Cabe ressaltar que uma pessoa não precisa estar associada a um computador, como é o caso do registro cujo CPF é "444".

Embora só tenhamos mostrado atributos em entidades até o momento, os relacionamentos também podem possuir atributos,

independente da cardinalidade. A figura 4.14 apresenta um exemplo de relacionamento de cardinalidade *1:n* com um atributo.



CPF	Nome	Sexo
111	Ana	F
222	Beto	M
333	Lia	F
444	Caio	M

Cod	Modelo	Marca	P_CPF	Data
1A	Notebook	Apple	111	01-Jan-2011
2B	Desktop	Dell	111	02-Fev-2012
3C	Desktop	IBM	333	03-Mar-2013
4D	Netbook	HP	222	04-Abr-2014

Figura 4.14: Relacionamento de cardinalidade 1:n com um atributo.

A figura 4.14 apresenta o mesmo modelo da figura 4.13; a única diferença consiste no acréscimo do atributo `Data` no relacionamento `Usa`. Nesse caso, ao mapear o modelo conceitual para o lógico, todas as atividades realizadas no mapeamento da figura 4.13 serão realizadas e o atributo `Data` irá para a mesma tabela da chave estrangeira que representa o relacionamento `Usa`.

Outra situação interessante do relacionamento de cardinalidade 1:n é o relacionamento identificador:

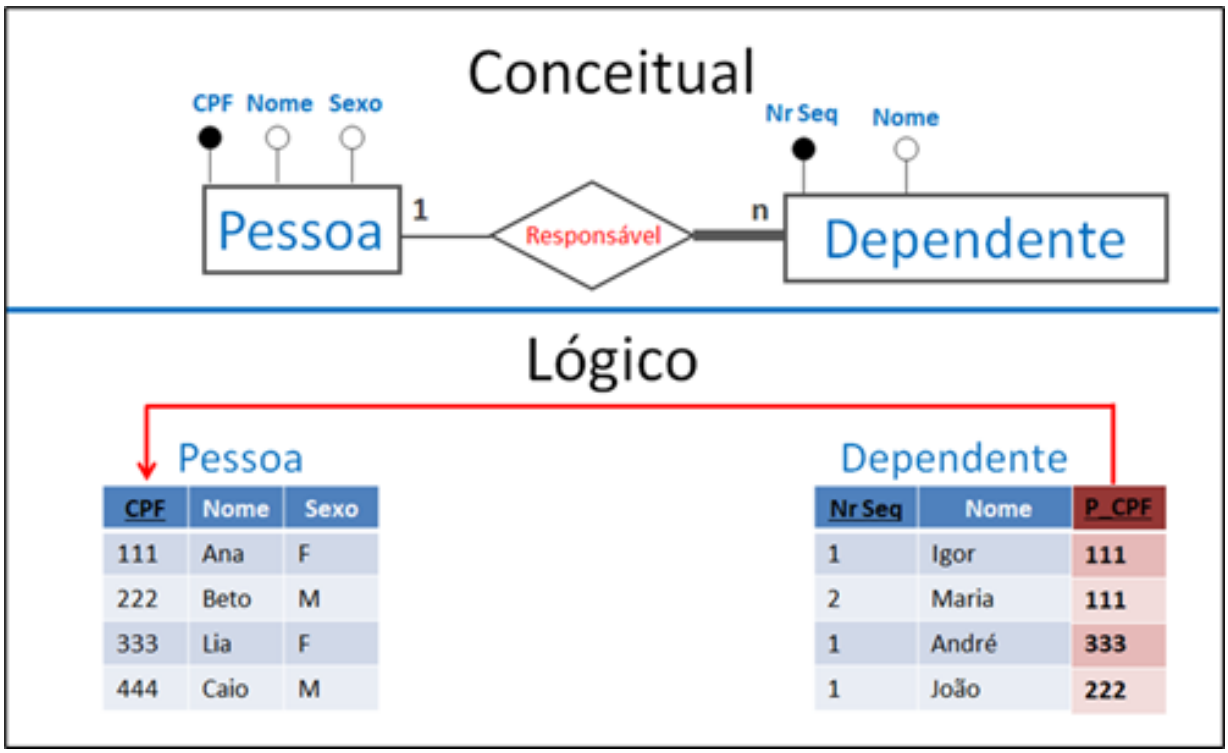


Figura 4.15: Relacionamento de cardinalidade 1:n identificador.

De acordo com a figura 4.15, uma `Pessoa` pode possuir vários `Dependentes`. O atributo `Nr Seq` na entidade `Dependente` representa o número de ordem de cada dependente em relação a uma pessoa. Por exemplo, `Igor` é o primeiro dependente de `Ana`, pois seu `Nr Seq` é "1"; `Maria` é a segunda dependente de `Ana`, e seu `Nr Seq` é "2"; `André` é o primeiro dependente de `Lia`, e seu `Nr Seq` é "1". Pode-se observar que o `Nr Seq` não é suficiente para diferenciar um registro dos demais na tabela `Dependente`.

O relacionamento identificador, representado pela linha mais grossa na figura 4.15, ocorre quando uma entidade possui uma chave parcial, o `Nr Seq` em `Dependente`, que não é suficiente para diferenciar um registro dos demais. Nesse caso, para que os registros da tabela `Dependente` possam ser diferenciados uns dos outros, é necessário criar uma chave primária composta pelos atributos `Nr Seq` e `P_CPF`, sendo que `P_CPF` é uma chave estrangeira que referencia a tabela `Pessoa`.

Assim, para que um `Dependente` exista, é necessário que haja uma `Pessoa` relacionada, pois a sua chave primária é composta pela chave estrangeira que referencia `Pessoa`. Nesse caso, a entidade `Dependente` é chamada de *entidade fraca*, pois sua existência depende de outra entidade, a entidade `Pessoa`, que é denominada *entidade forte* ou *entidade identificadora*.

Cardinalidade n:n

Os relacionamentos com cardinalidade *n:n* também podem ser opcionais ou obrigatórios. A obrigatoriedade desses relacionamentos é implementada por meio de programação, por isso abordaremos apenas os opcionais. A figura 4.16 apresenta um exemplo de relacionamento de cardinalidade *n:n*.

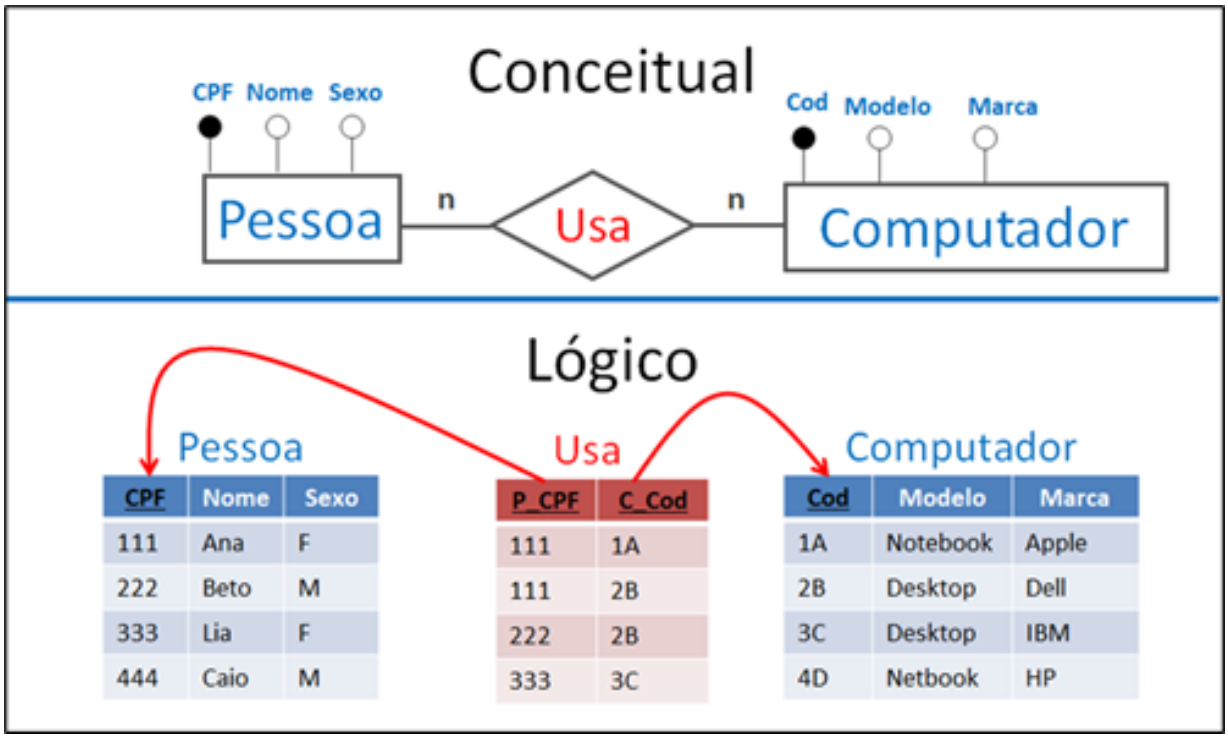
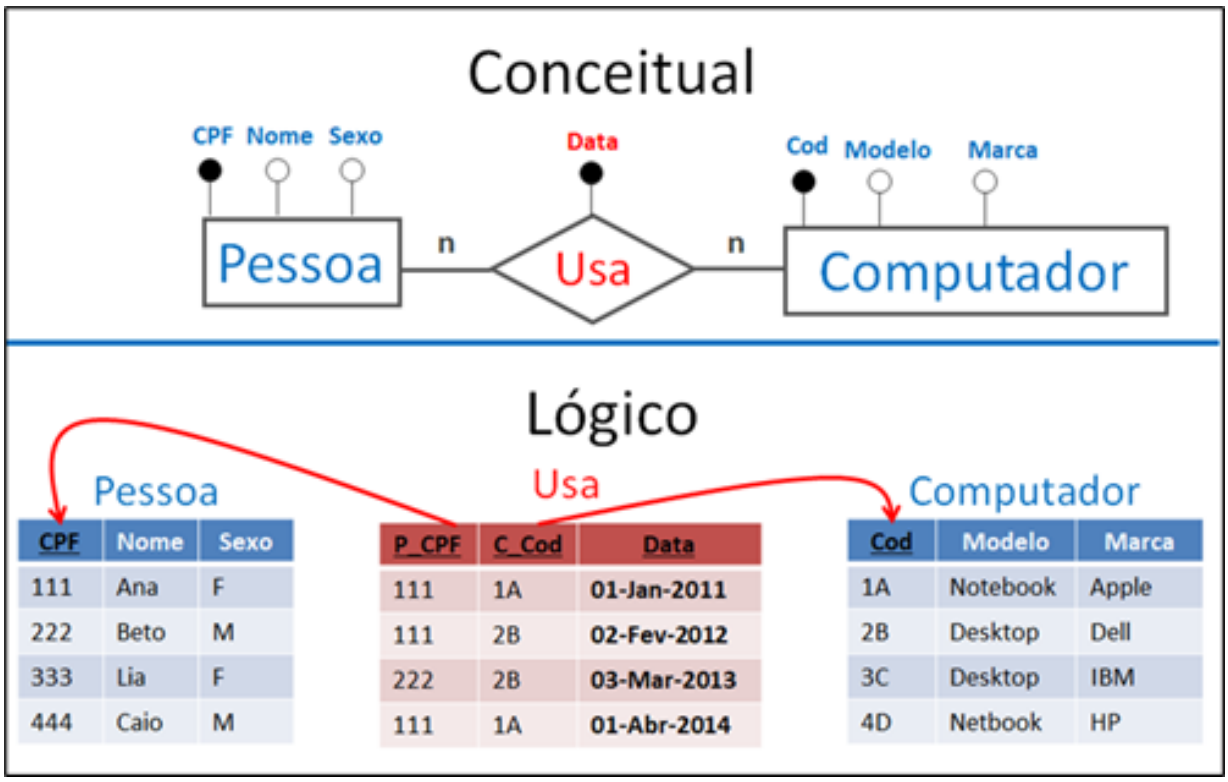


Figura 4.16: Relacionamento de cardinalidade n:n.

Como pode ser observado na figura 4.16, uma pessoa pode estar relacionada a nenhum ou vários computadores, e um computador pode estar relacionado a nenhuma ou várias pessoas. Nesse caso,

ao mapear o modelo conceitual para o lógico, o relacionamento `Usa` de cardinalidade *n:n* foi mapeado para uma tabela à parte denominada `Usa` com duas colunas, `P_CPF` e `C_Cod`. A coluna `P_CPF` é uma chave estrangeira que referencia `CPF` da tabela `Pessoa`, e `C_Cod` é uma chave estrangeira que referencia `Cod` da tabela `Computador`. A chave primária da tabela `Usa` é composta pelas colunas `P_CPF` e `C_Cod`.

Em relacionamento de cardinalidade n:n podem existir situações mais complexas do que a apresentada na figura 4.16. Por exemplo: além de uma pessoa poder estar relacionada a nenhum ou vários computadores e um computador poder estar relacionado a nenhuma ou várias pessoas, a mesma pessoa poder estar associada ao mesmo computador várias vezes em momentos diferentes. Nesse caso, há a necessidade de se criar um atributo identificador `Data` no relacionamento `Usa`. O atributo `Data` comporá a chave primária da tabela `Usa` no nível lógico, que será composta pelos atributos `P_CPF`, `C_Cod` e `Data`, conforme apresentado na figura 4.17.



CPF	Nome	Sexo
111	Ana	F
222	Beto	M
333	Lia	F
444	Caio	M

P_CPF	C_Cod	Data
111	1A	01-Jan-2011
111	2B	02-Fev-2012
222	2B	03-Mar-2013
111	1A	01-Abr-2014

Cod	Modelo	Marca
1A	Notebook	Apple
2B	Desktop	Dell
3C	Desktop	IBM
4D	Netbook	HP

Figura 4.17: Relacionamento de cardinalidade n:n com atributo identificador.

Conforme a figura 4.17, a pessoa cujo `CPF` é "111" se relaciona duas vezes com o mesmo computador, cujo `Cod` é "1A". Os registros na tabela `Usa` com `P_CPF` "111", `C_Cod` "1A" e `Data` "01-Jan-2011"; e `P_CPF` "111", `C_Cod` "1A" e `Data` "01-Abr-2014" materializam esse relacionamento. Pode-se perceber que, se a data não fizesse parte da chave primária da tabela `Usa`, não haveria a garantia da unicidade desses dois registros.

Autorrelacionamento

Nas seções anteriores, apresentamos as cardinalidades de relacionamentos binários entre duas entidades distintas, porém esses relacionamentos podem ocorrer de forma recursiva, ou seja, uma entidade pode se relacionar com ela mesma. Esse tipo de relacionamento é denominado **autorrelacionamentos**. As figuras 4.18, 4.19 e 4.20 apresentam as cardinalidades desses

relacionamentos no nível conceitual e como eles são mapeados para o nível lógico.

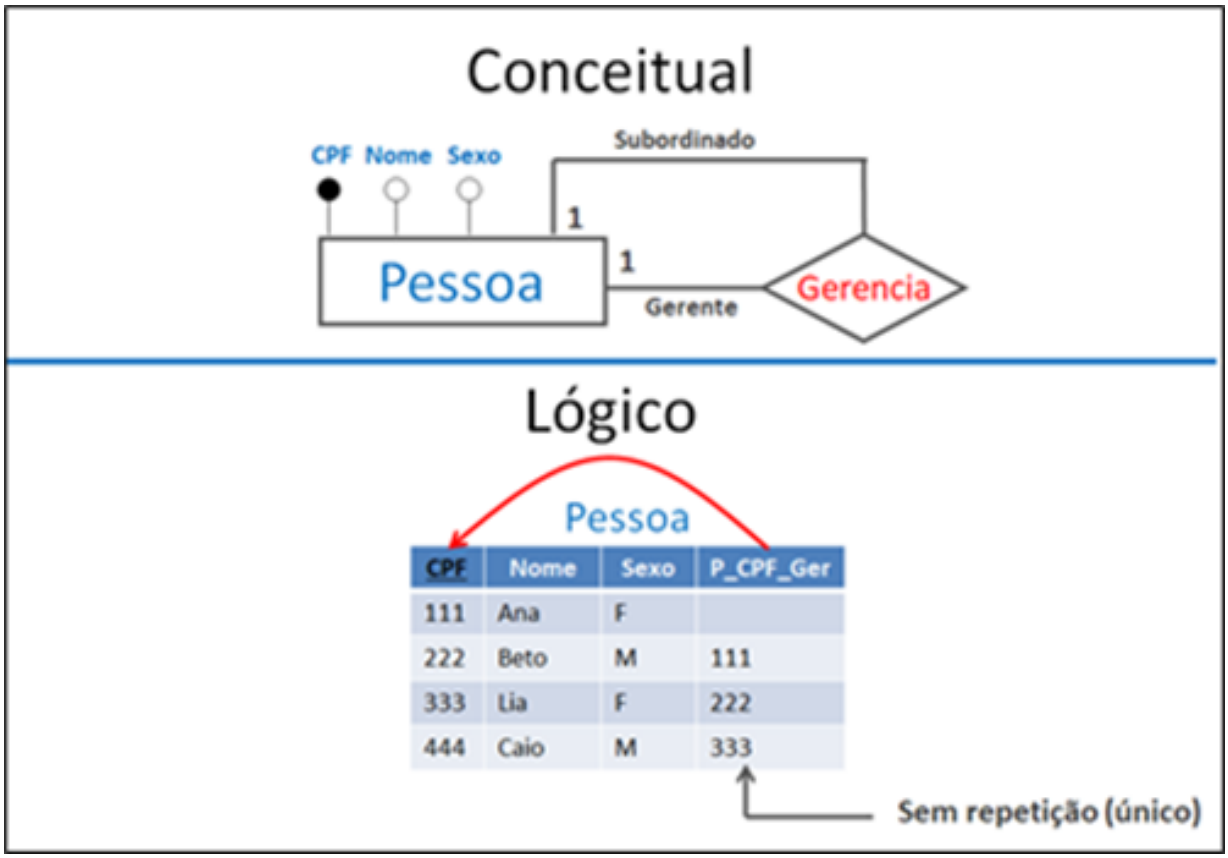


Figura 4.18: Autorrelacionamento de cardinalidade 1:1.

A figura 4.18 apresenta o autorrelacionamento de cardinalidade 1:1 denominado `Gerencia` da entidade `Pessoa`. Uma pessoa pode gerenciar zero ou uma pessoa, e pode ser gerenciada por zero ou uma pessoa. Esse relacionamento foi mapeado para a coluna `P_CPF_Ger` da tabela `Pessoa`. A coluna `P_CPF_Ger`, que representa o CPF do gerente, também é uma chave estrangeira que referencia a coluna `CPF` da mesma tabela. Um exemplo de leitura de um registro seria: o `P_CPF_Ger` cujo valor é "111" gerencia o `CPF` "222", ou seja, "Ana" gerencia "Beto". Para que esse relacionamento tenha cardinalidade 1:1, é necessário que a coluna `P_CPF_Ger` seja única, ou seja, não possua valores repetidos; caso contrário, a cardinalidade seria 1:n.

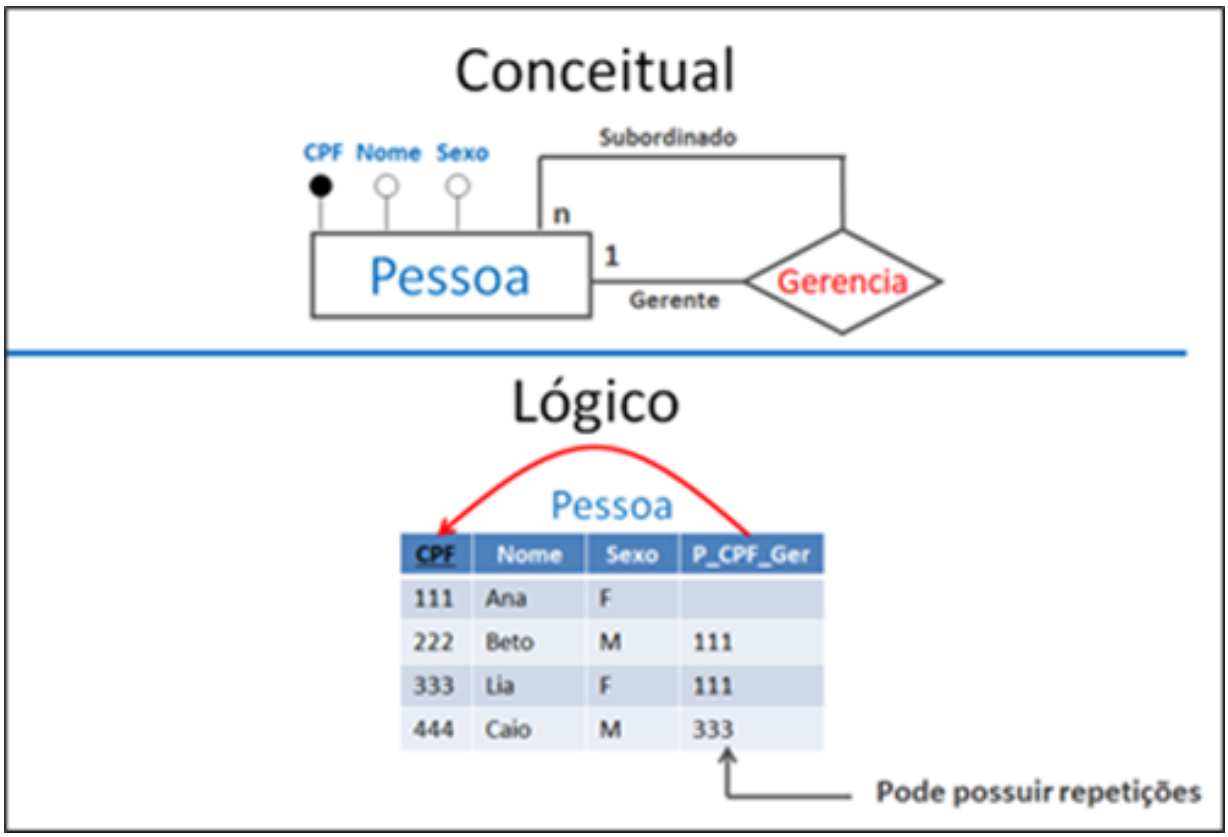


Figura 4.19: Autorrelacionamento de cardinalidade 1:n.

A figura 4.19 apresenta o autorrelacionamento `Gerencia` de cardinalidade 1:n da entidade `Pessoa`. Uma pessoa pode gerenciar zero ou várias pessoas, e pode ser gerenciada por zero ou uma pessoa. Esse relacionamento foi mapeado para a coluna `P_CPF_Ger` da tabela `Pessoa`. A coluna `P_CPF_Ger` também é uma chave estrangeira que referencia a coluna `CPF` da mesma tabela. Nesse caso, a coluna `P_CPF_Ger` pode possuir repetições, pois a cardinalidade é 1:n. Na figura, a pessoa cujo `CPF` é "111" gerencia as pessoas com os CPFs "222" e "333", ou seja, "Ana" gerencia "Beto" e "Lia".

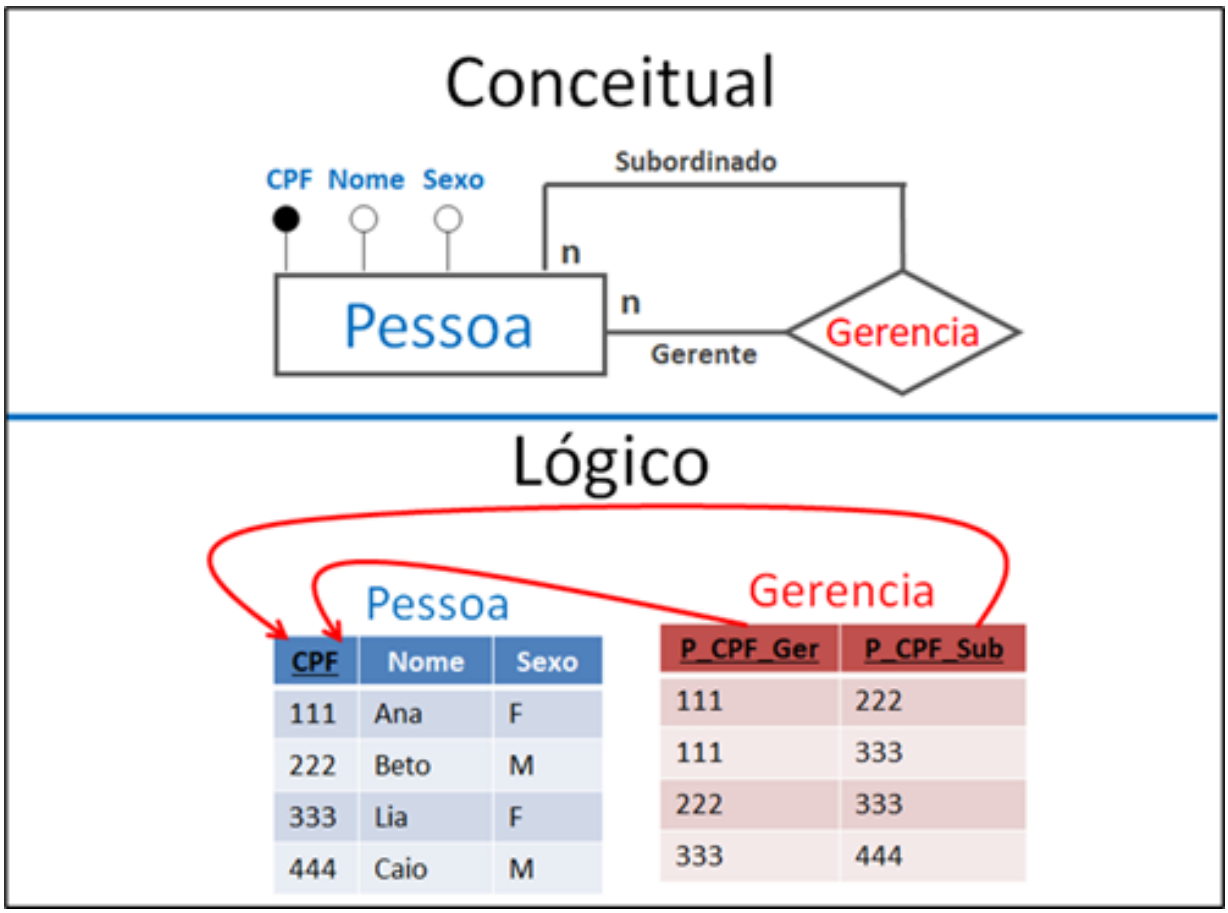


Figura 4.20: Autorrelacionamento de cardinalidade n:n.

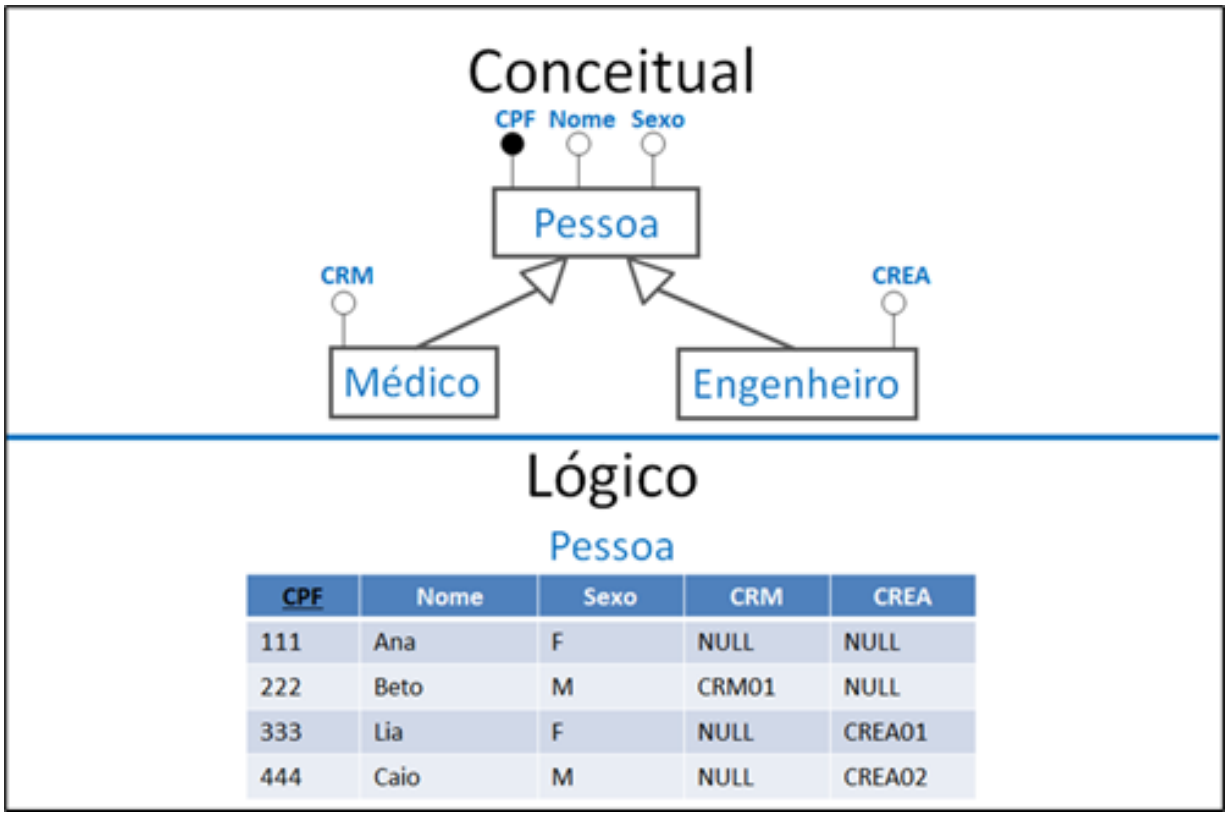
A figura 4.20 apresenta o autorrelacionamento `Gerencia` de cardinalidade n:n da entidade `Pessoa`. Uma pessoa pode gerenciar zero ou várias pessoas, e pode ser gerenciada por zero ou várias pessoas.

Esse relacionamento foi mapeado para uma tabela à parte, denominada `Gerencia`, com as colunas `P_CPF_Ger` e `P_CPF_Sub`, que representam o CPF da pessoa gerente e da gerenciada, respectivamente. Essas duas colunas compõem a chave primária da tabela, e cada uma delas é uma chave estrangeira que referencia a coluna `CPF` da tabela Pessoa. Nesse caso, ambas as colunas da tabela `Gerencia` podem possuir repetições, pois a cardinalidade é n:n.

Na figura, a pessoa cujo CPF é "111" gerencia as pessoas com os CPFs "222" e "333", e a pessoa cujo CPF é "333" é gerenciada pelas pessoas com os CPFs "111" e "222", ou seja, "Ana" gerencia "Beto" e "Lia"; e "Lia" é gerenciada por "Ana" e "Beto".

Generalização/especialização (Herança)

O relacionamento de generalização/especialização ou herança pode ser mapeado de três formas diferentes a partir do modelo conceitual para o modelo lógico. As figuras 4.21, 4.22 e 4.23 apresentam essas formas.



CPF	Nome	Sexo	CRM	CREA
111	Ana	F	NULL	NULL
222	Beto	M	CRM01	NULL
333	Lia	F	NULL	CREA01
444	Caio	M	NULL	CREA02

Figura 4.21: Relacionamento de herança mapeado para uma tabela.

A figura 4.21 apresenta o mapeamento do relacionamento de herança no modelo conceitual contendo três entidades para uma tabela no modelo lógico. Nessa figura, pode-se observar que os atributos das entidades `Pessoa`, `Médico` e `Engenheiro` foram mapeados para a tabela `Pessoa`, cuja chave primária é o campo `CPF`.

A diferenciação do tipo de pessoa armazenada na tabela pode ser feita observando os valores dos campos `CRM` e `CREA`.

Levando em conta os valores desses campos, pode-se afirmar que "Ana" é do tipo `Pessoa`, pois os campos `CRM` e `CREA` desse registro são nulos; "Beto" é médico, pois o campo `CREA` é nulo e o campo `CRM` é valorado; e "Lia" e "Caio" são engenheiros, pois os campos `CRM` desses registros são nulos e os campos `CREA` são valorados.

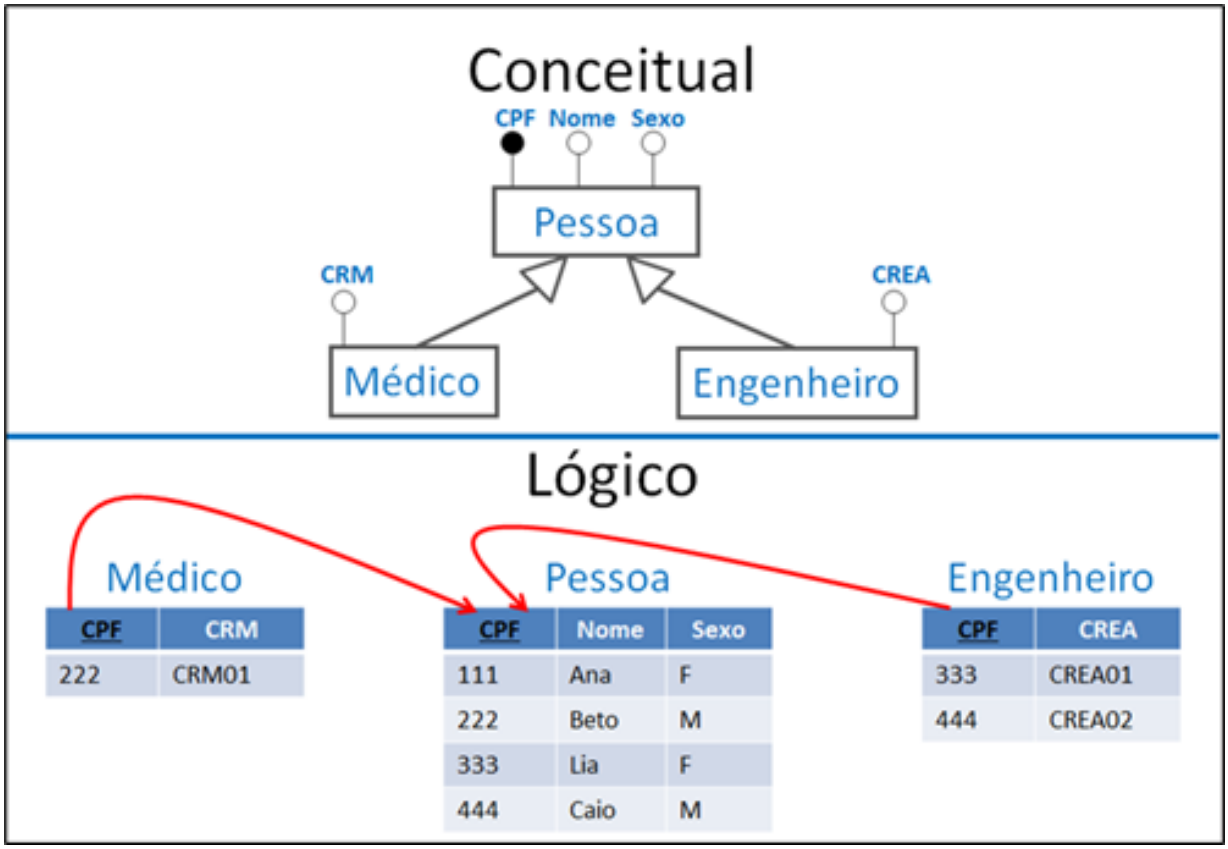


Figura 4.22: Relacionamento de herança mapeado para três tabelas.

A figura 4.22 apresenta o mapeamento do relacionamento de herança no modelo conceitual contendo três entidades para três tabelas no modelo lógico. Nessa figura, pode-se observar que os atributos da entidade `Pessoa` foram mapeados para a tabela `Pessoa` cuja chave primária é o campo `CPF`; o atributo da entidade `Medico` foi mapeado para a tabela `Medico` cuja chave primária é o campo `CPF`, que também é chave estrangeira e referencia `CPF` da tabela

`Pessoa` ; e o atributo da entidade `Engenheiro` foi mapeado para a tabela `Engenheiro` , cuja chave primária é o campo `CPF` , que também é chave estrangeira e referencia `CPF` da tabela `Pessoa` . Cada tipo do modelo conceitual está armazenado em uma tabela distinta no modelo lógico.

A restrição de integridade referencial criada entre os atributos `CPF` das tabelas `Medico` e `Engenheiro` em relação ao atributo `CPF` da tabela `Pessoa` exige que todo engenheiro e médico inserido nessas tabelas existam como pessoas na tabela `Pessoa` , o que garante a consistência dos dados.

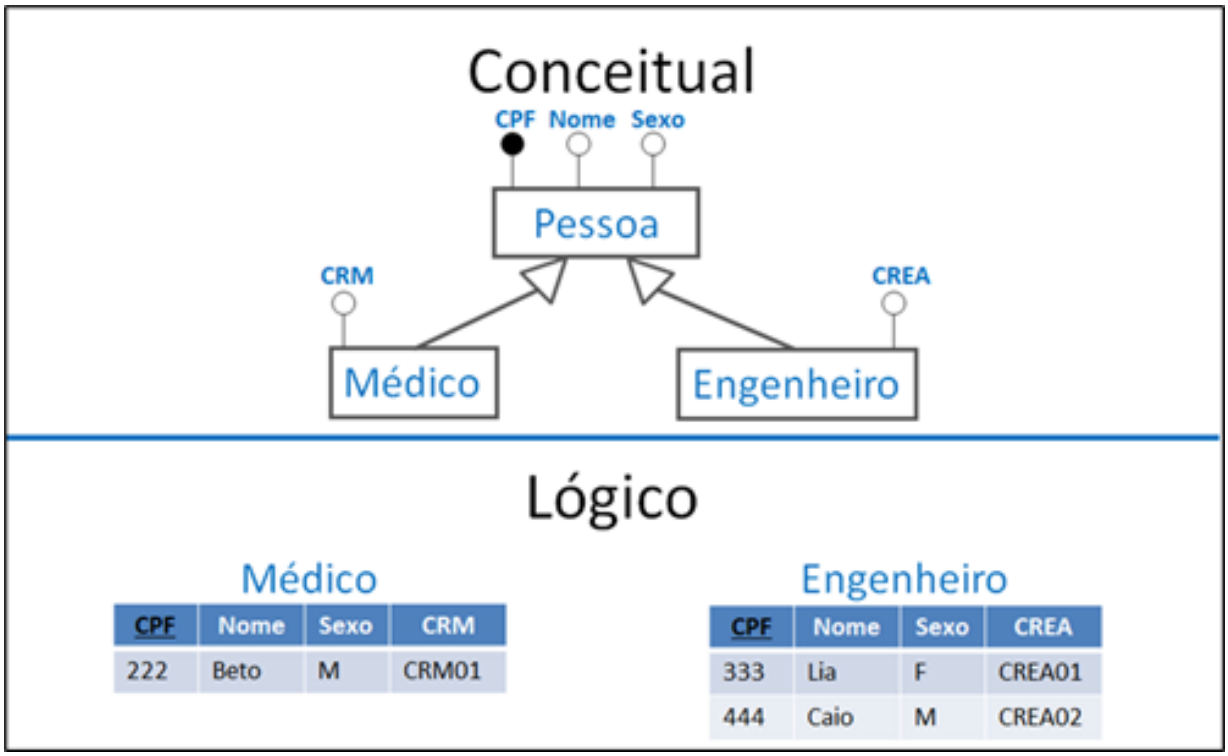


Figura 4.23: Relacionamento de herança mapeado para duas tabelas.

A figura 4.23 apresenta o mapeamento do relacionamento de herança no modelo conceitual contendo três entidades para duas tabelas no modelo lógico. Nessa figura, pode-se observar que os atributos das entidades `Pessoa` e `Medico` foram mapeados para a

tabela `Medico`, cuja chave primária é o campo `CPF`, e os atributos das entidades `Pessoa` e `Engenheiro` foram mapeados para a tabela `Engenheiro`, cuja chave primária é o campo `CPF`.

Essa estratégia de mapeamento possui uma limitação:, observe que, se no domínio observado houver uma instância que seja apenas do tipo `Pessoa`, como era o caso do registro de `CPF` `111" nos mapeamentos anteriores, essa instância não poderá ser armazenada nas tabelas geradas, ou seja, esse mapeamento só permite armazenar dados de `Médico` e `Engenheiro`, por isso o uso dessa estratégia deve levar em conta essa peculiaridade.

Relacionamentos ternários

Nas seções anteriores, abordamos os modelos conceitual e lógico apresentando apenas relacionamentos de grau dois, ou seja, relacionamentos binários. O grau do relacionamento corresponde ao número de entidades envolvidas na associação.

No entanto, é muito comum os modelos conceituais possuírem relacionamentos de graus superiores a dois. A figura 4.24 apresenta um exemplo de relacionamento de grau três, ou seja, ternário.

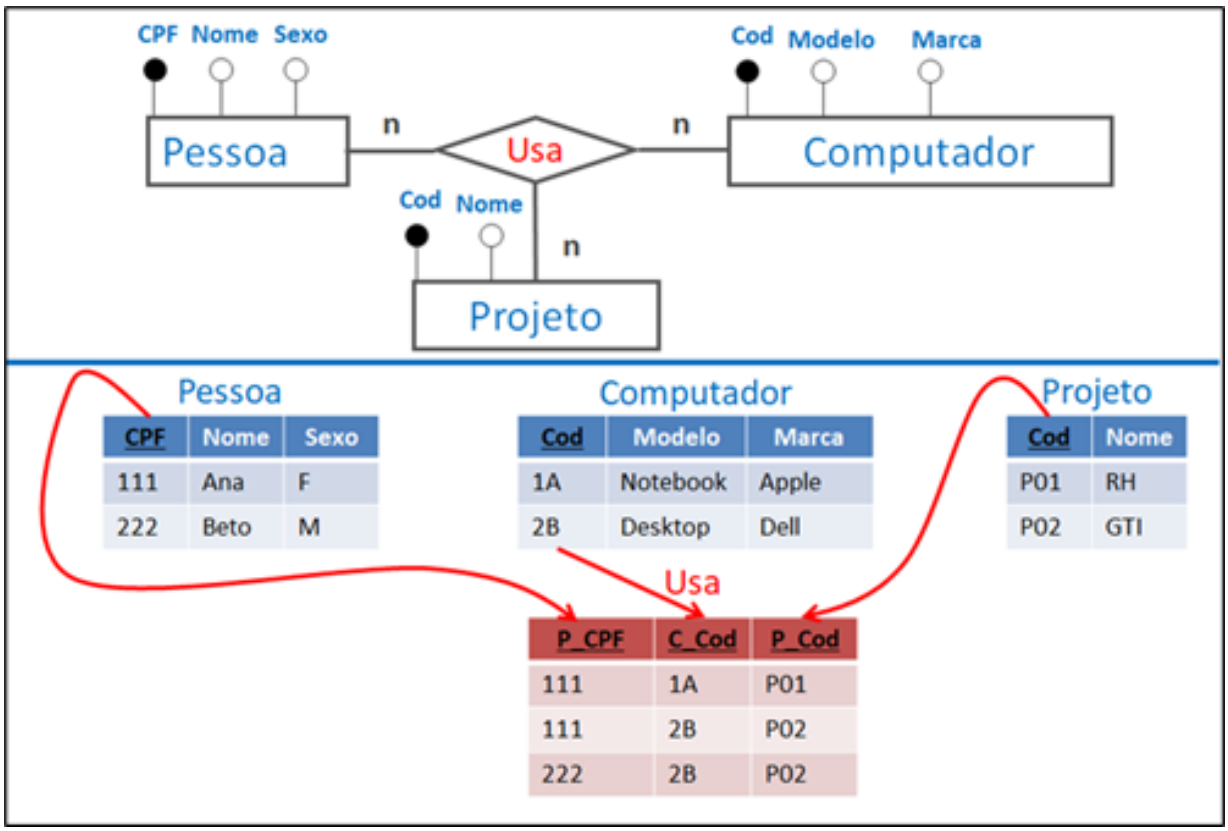


Figura 4.24: Relacionamento ternário.

Como pode ser observado na figura 4.24, o relacionamento `Usa` envolve três entidades, `Pessoa`, `Computador` e `Projeto`. Nesse caso, ao mapear o modelo conceitual para o lógico, o relacionamento `Usa` foi mapeado para uma tabela à parte denominada `Usa` com três colunas, `P_CPF`, `C_Cod` e `P_Cod`. A coluna `P_CPF` é uma chave estrangeira que referencia `CPF` da tabela `Pessoa`; `C_Cod` é uma chave estrangeira que referencia `Cod da` tabela `Computador`; e `P_Cod` é uma chave estrangeira que referencia Cod da tabela `Projeto`. A chave primária da tabela `Usa` é composta pelas colunas `P_CPF`, `C_Cod` e `P_Cod`.

Especificação de Banco de Dados Relacional

Na seção anterior, para mapear o modelo conceitual para o lógico relacional, por questões de didática, desenhamos as tabelas resultantes desse mapeamento. No entanto, apesar de essa

estratégia facilitar o aprendizado, na prática costumamos representar o modelo lógico relacional por meio da especificação do banco de dados.

A especificação do Banco de Dados Relacional (BDR), ou esquema do BDR, deve conter no mínimo:

- Tabelas que formam o BDR;
- Colunas que as tabelas possuem;
- Restrições de integridade.

As regras para a especificação do BDR são:

- Cada tabela é listada pelo seu nome;
- Após o nome da tabela, são listados entre parênteses os nomes das colunas, separados por vírgula;
- A coluna ou colunas que contêm a chave primária são sublinhadas;
- Após a definição da tabela, são listadas as definições das chaves estrangeiras que aparecem na tabela;
- Se a chave estrangeira é formada por várias colunas, elas são declaradas separadas por vírgulas.

A figura 4.25 apresenta duas tabelas com as respectivas especificações.

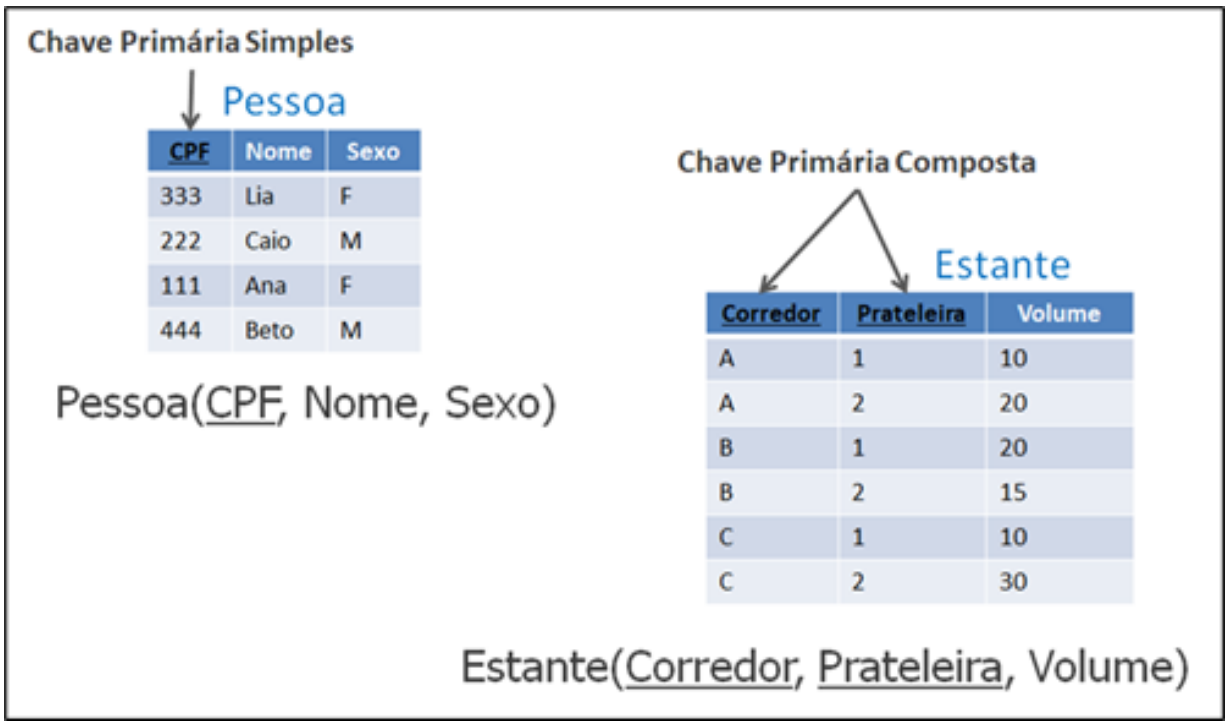


Figura 4.25: Especificação do BDR.

Na figura 4.25, a primeira tabela é a tabela `Pessoa`, que possui as colunas `CPF`, `Nome` e `Sexo`, e `CPF` é a chave primária da tabela. Abaixo da tabela `Pessoa` está a sua especificação, iniciada com o termo "Pessoa", que é o nome da tabela, seguido pelos termos que representam as colunas entre parênteses, que são "CPF", "Nome" e "Sexo". O termo "CPF" está sublinhado por ser a chave primária da tabela.

A outra tabela da figura 4.25 é a tabela `Estante`, que possui as colunas `Corredor`, `Prateleira` e `Volume`, e a chave primária da tabela é composta pelas colunas `Corredor` e `Prateleira`. Abaixo da tabela `Estante`, está a sua especificação iniciada com o termo "Estante", que é o nome da tabela, seguido pelos termos que representam as colunas entre parênteses, que são "Corredor", "Prateleira" e "Volume". Os termos "Corredor" e "Prateleira" estão sublinhados por comporem a chave primária da tabela.

A figura 4.26 apresenta a mesma tabela `Pessoa` da figura 4.25 relacionada à tabela `Computador` por meio do relacionamento `Usa` de

cardinalidade 1:n.

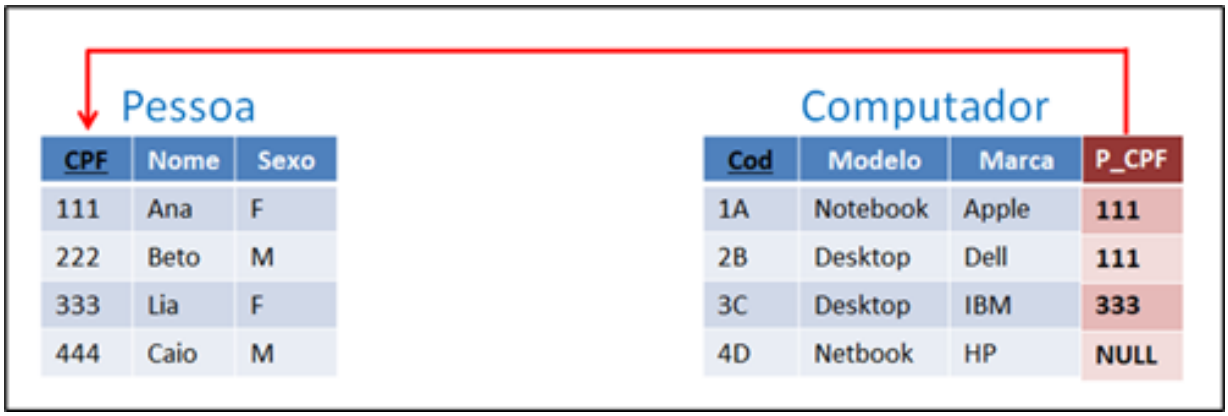


Figura 4.26: Modelo lógico com cardinalidade de relacionamento 1:n.

A especificação da tabela `Computador` da figura 4.26 é a seguinte:

- Computador(Cod, Modelo, Marca, P_CPF)
- P_CPF referencia Pessoa(CPF)

A especificação inicia com o termo "Computador", que é o nome da tabela, seguido pelos termos que representam as colunas entre parênteses, que são "Cod", "Modelo", "Marca" e "P_CPF". O termo "Cod" está sublinhado por ser a chave primária da tabela. A segunda linha da especificação lista a definição da chave estrangeira que aparece na tabela, que é a coluna `P_CPF`, que referencia a coluna `CPF` da tabela `Pessoa`.

A figura 4.27 apresenta a tabela `Pessoa` relacionada à tabela `Computador` por meio do relacionamento `Usa` de cardinalidade n:n.

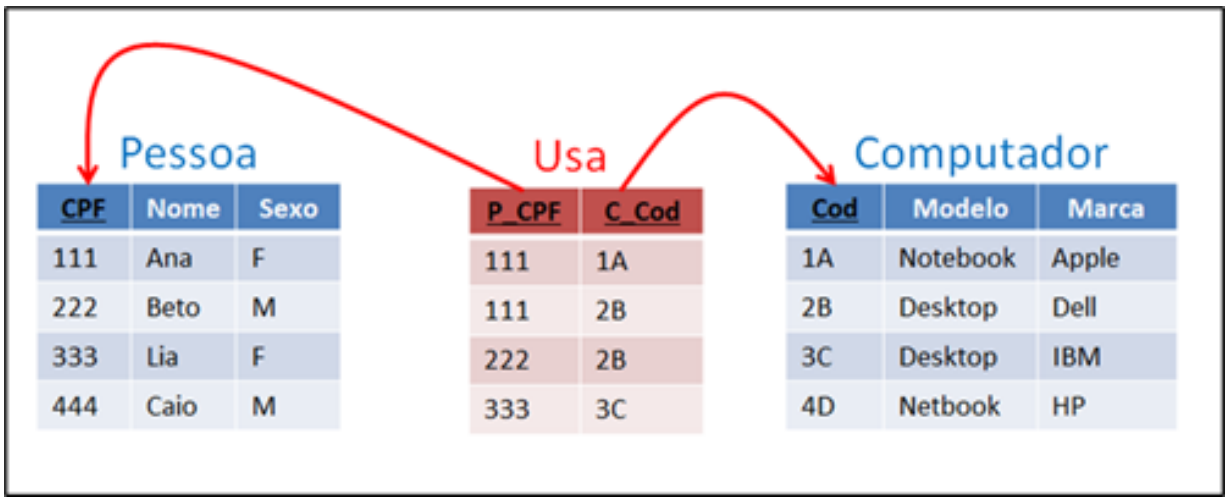


Figura 4.27: Modelo lógico com cardinalidade de relacionamento n:n.

A especificação das três tabelas da figura 4.27 é a seguinte:

- Pessoa(CPF, Nome, Sexo)
- Computador(Cod, Modelo, Marca)
- Usa(P_CPF, C_Cod)
- P_CPF referencia Pessoa(CPF)
- C_Cod referencia Computador(Cod)

Na primeira linha, a especificação inicia com o termo "Pessoa", que é o nome da tabela, seguido pelos termos que representam as colunas entre parênteses, que são "CPF", "Nome" e "Sexo". O termo "CPF" está sublinhado por ser a chave primária da tabela. A segunda linha inicia com o termo "Computador", que é o nome da tabela, seguido pelos termos que representam as colunas entre parênteses, que são "Cod", "Modelo" e "Marca". O termo "Cod" está sublinhado por ser a chave primária da tabela. A terceira linha inicia com o termo "Usa", que é o nome da tabela, seguido pelos termos que representam as colunas entre parênteses, que são "P_CPF" e "C_Cod", sublinhados por comporem a chave primária da tabela. A quarta e quinta linhas da especificação listam as definições das chaves estrangeiras: a coluna `P_CPF`, que referencia a coluna `CPF` da tabela `Pessoa`, e `C_Cod`, que referencia a coluna `Cod` da tabela Computador.

4.3 Normalização

A **normalização** é um recurso empregado **durante o projeto lógico** de um banco de dados a fim de evitar redundâncias acidentais (não controladas) e anomalias de inserção, exclusão e atualização dos dados. Trata-se de um mecanismo formal para analisar modelos lógicos baseado nas suas chaves e nas dependências funcionais entre seus atributos. Normalmente, um projeto conceitual bem-feito resulta naturalmente em esquemas lógicos normalizados.

Embora existam pelo menos cinco formas normais, abordaremos apenas as três primeiras, pois elas já são suficientes para a geração de modelos lógicos com qualidade.

Primeira Forma Normal (1FN)

A primeira forma normal faz parte da definição formal de tabela, ou seja, uma estrutura para ser considerada uma tabela, precisa respeitar a primeira forma normal. Essa forma normal foi definida para não permitir atributos multivalorados, atributos compostos e suas combinações. Um campo é considerado *composto* quando seu conteúdo pode ser subdividido em conceitos distintos, e *multivalorado* quando possui mais de um valor.

A definição para 1FN é: **Uma relação está na 1FN se, e somente se, todos os seus atributos contêm apenas valores atômicos (simples, indivisíveis)**.

A figura 4.28 apresenta a normalização da tabela `Pessoa` para a 1FN.

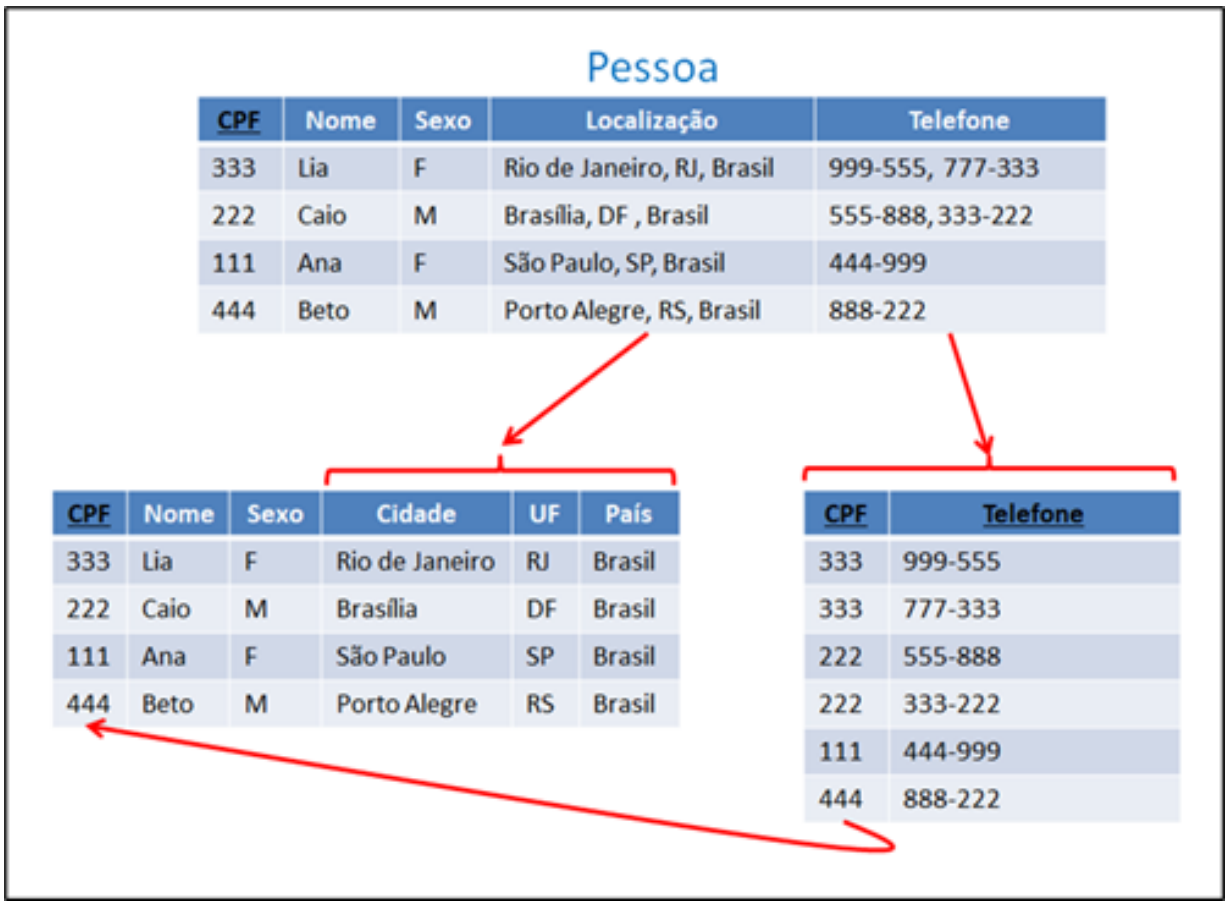


Figura 4.28: Normalização da tabela Pessoa para a 1FN.

A tabela `Pessoa` apresentada na parte superior da figura 4.28 não está na 1FN, pois o campo `Localização` é composto (possui mais de um conceito) e o campo `Telefone` é multivalorado (possui mais de uma ocorrência). Para resolver esses problemas, a tabela `Pessoa` deve ser decomposta em outras duas, que são apresentadas na parte inferior da mesma figura.

O campo `Localização` foi subdivido nos campos `Cidade`, `UF` e `País`, por serem conceitos distintos. Já o campo `Telefone`, que possui várias ocorrências para uma mesma tupla, foi retirado da tabela `Pessoa` e transferido para uma nova tabela composta pelas colunas `CPF` e `Telefone`, cuja chave primária é formada por essas duas colunas, o `CPF` também é chave estrangeira. Assim, cada tupla passa a possuir apenas um telefone, cuja unicidade é garantida pela combinação ds colunas `CPF` com o `Telefone`.

Segunda Forma Normal (2FN)

A definição para 2FN é: **Uma relação está na 2FN se, e somente se, estiver na 1FN e não contiver dependências parciais**.

A **dependência parcial** ocorre quando uma coluna depende apenas de uma parte de uma chave primária composta. Assim, para estar na 2FN, uma relação não deve possuir atributo não-chave funcionalmente determinado por parte da chave primária.

A figura 4.29 apresenta a tabela `Empregado_Projeto`, que representa os empregados de uma organização e os projetos em que eles são alocados. Essa tabela não respeita a segunda forma normal.

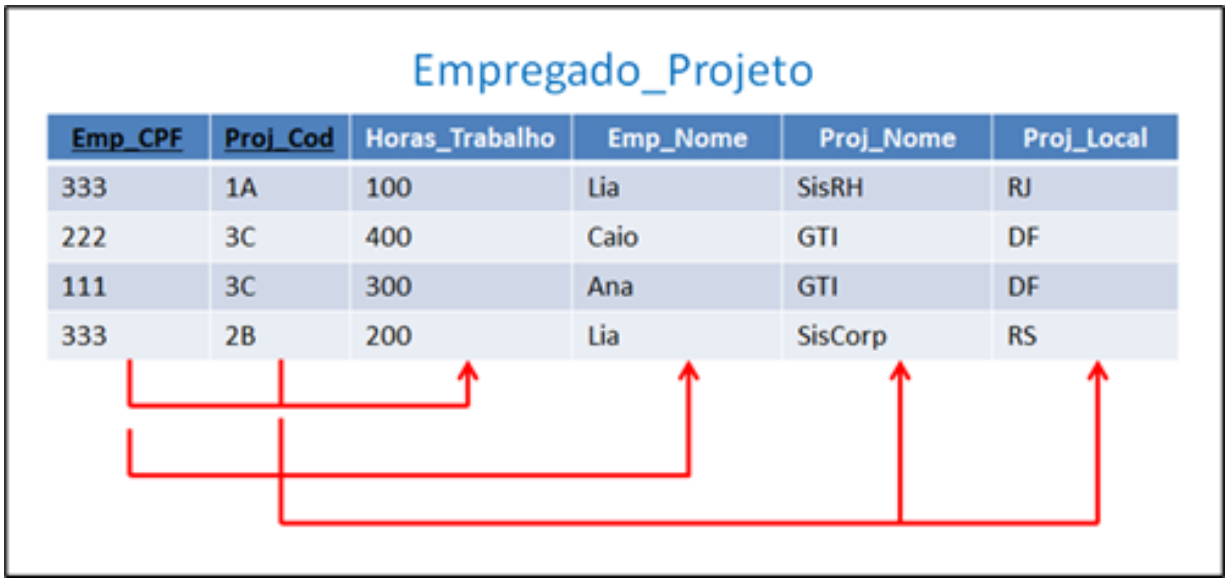


Figura 4.29: Tabela Empregado_Projeto.

A tabela `Empregado_Projeto`, apresentada na figura 4.29, possui uma chave primária composta, formada pelas colunas `Emp_CPF` e `Proj_Cod`, que representam o `CPF` dos empregados e o código dos projetos em que estão alocados, respectivamente. O campo `Horas_Trabalho` representa as horas que um empregado trabalha em um projeto. Esse campo é determinado pela chave completa, ou seja, para saber o número de horas trabalhadas de um empregado,

é necessário saber o seu CPF (campo `Emp_CPF`) e o projeto em que trabalhou (campo `Proj_Cod`).

No entanto, o nome do empregado (campo `Emp_Nome`) é determinado apenas pelo CPF (campo `Emp_CPF`), pois é independente do projeto (campo `Proj_Cod`) de que ele participa, causando uma dependência parcial, o que impede a tabela de atingir a 2FN. Outras dependências parciais ocorrem em relação aos campos `Proj_Nome` e `Proj_Local`, pois o nome e o local dos projetos dependem apenas do código do projeto (campo `Proj_Cod`), não dependem do CPF do empregado.

A ausência da segunda forma normal pode causar anomalias de inserção, exclusão e atualização dos registros da tabela, conforme apresentado nas figuras 4.30, 4.31 e 4.32.

Empregado_Projeto

Emp_CPF	Proj_Cod	Horas_Trabalho	Emp_Nome	Proj_Nome	Proj_Local
333	1A	100	Lia	SisRH	RJ
222	3C	400	Caio	GTI	DF
111	3C	300	Ana	GTI	DF
333	2B	200	Lia	SisCorp	RS
555	1A	50	Igor	SisRH	RJ

Figura 4.30: Anomalia de inserção na tabela Empregado_Projeto.

De acordo com a figura 4.30, para inserir um empregado na tabela `Empregado_Projeto`, é necessário inserir todos os dados do projeto em que ele trabalha. O correto seria inserir apenas os dados do empregado, independentemente do projeto.

Na figura, para inserir o empregado cujo CPF é "555", é necessário inserir todos os dados do projeto "1A". Essa anomalia de inserção pode causar um outro problema: pode gerar dois registros do

mesmo projeto com dados diferentes, como o nome ou o local, por exemplo.

Empregado_Projeto

Emp_CPF	Proj_Cod	Horas_Trabalho	Emp_Nome	Proj_Nome	Proj_Local
333	1A	100	Lia	SisRH	RJ
222	3C	400	Caio	GTI	DF
111	3C	300	Ana	GTI	DF
333	2B	200	Lia	SisCorp	RS

Figura 4.31: Anomalia de exclusão na tabela Empregado_Projeto.

De acordo com a figura 4.31, ao excluir um empregado da tabela `Empregado_Projeto` , caso esse empregado seja o único em determinado projeto, o projeto deixa de existir. Essa é uma anomalia que ocorre durante a exclusão de registros.

Na figura, ao deletar o empregado cujo CPF é "333", o projeto "2B" deixa de existir na base de dados.

Empregado_Projeto

Emp_CPF	Proj_Cod	Horas_Trabalho	Emp_Nome	Proj_Nome	Proj_Local
333	1A	100	Lia	SisRH	RJ
222	3C	400	Caio	GTI	DF
111	3C	300	Ana	GTI	DF
333	2B	200	Lia	SisCorp	RS

Figura 4.32: Anomalia de atualização na tabela Empregado_Projeto.

De acordo com a figura 4.32, para atualizar os dados de um projeto na tabela `Empregado_Projeto`, é necessário atualizar todos os registros dos empregados alocados a esse projeto. O correto seria atualizar os dados do projeto apenas no registro que representa o projeto.

Na figura, para atualizar os dados do projeto "3C", é necessário atualizar os registros dos empregados com os CPFs "111" e "222", por estarem alocados a esse projeto. Essa anomalia de atualização pode causar um outro problema: pode gerar dois registros do mesmo projeto com dados diferentes (por exemplo, o nome ou o local), assim como ocorre na inserção.

A figura 4.33 apresenta a normalização da tabela Empregado_Projeto para a 2FN.

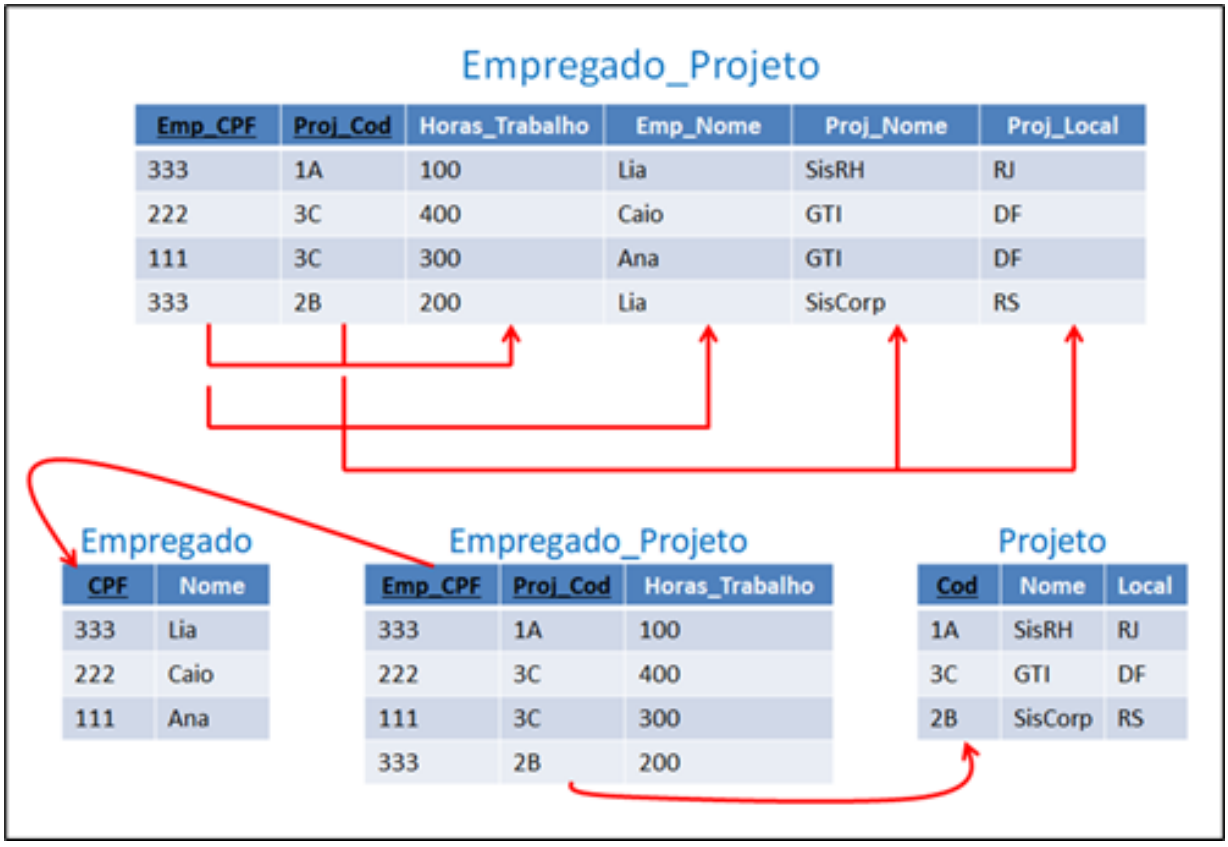


Figura 4.33: Normalização da tabela Empregado_Projeto para a 2FN.

A tabela `Empregado_Projeto` apresentada na parte superior da figura 4.33 não está na 2FN, pois os campos `Emp_Nome`, `Proj_Nome` e `Proj_Local` apresentam dependência parcial em relação à chave primária composta pelos campos `Emp_CPF` e `Proj_Cod`.

Para atingir a 2FN, a tabela `Empregado_Projeto` foi transformada em três tabelas: a própria tabela `Empregado_Projeto`, a `Empregado` e a `Projeto`, conforme apresentado na parte inferior da figura. Com essa nova estrutura, é possível atingir a 2FN e eliminar as anomalias de inserção, exclusão e atualização.

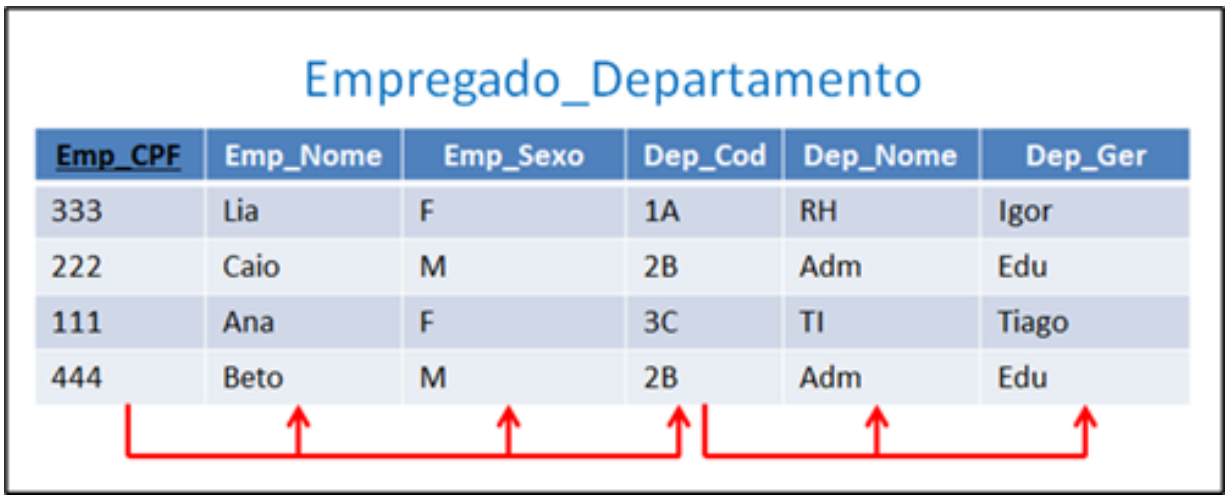
Na nova estrutura, para inserir um novo empregado, basta criar um registro na tabela `Empregado`. Não é necessário inserir os dados de projeto, pois eles agora são armazenados em uma tabela à parte. Para excluir um empregado, basta deletar o registro correspondente da tabela `Empregado` sem interferir na existência de qualquer projeto. Para atualizar os dados de um projeto, basta atualizar apenas um registro da tabela `Projeto`. Finalmente, para excluir um empregado de projeto, basta excluir o registro equivalente da tabela `Empregado_Projeto`, o que não causará impacto nas tabelas Empregado e Projeto.

Terceira Forma Normal (3FN)

A definição para 3FN é: **Uma relação está em 3FN se, e somente se, estiver na 2FN e nenhum atributo não-primo (isto é, que não seja membro de uma chave) for transitivamente dependente da chave primária**.

A **dependência transitiva** ocorre quando uma coluna, além de depender da chave primária de uma tabela, depende de outra coluna ou conjunto de colunas da tabela que não fazem parte da chave primária. Para estar na 3FN, uma relação não deve ter um atributo não-chave funcionalmente determinado por um outro atributo não-chave.

A figura 4.34 apresenta a tabela `Empregado_Departamento`, que representa os empregados de uma organização e os departamentos em que eles são alocados. Essa tabela não respeita a terceira forma normal.



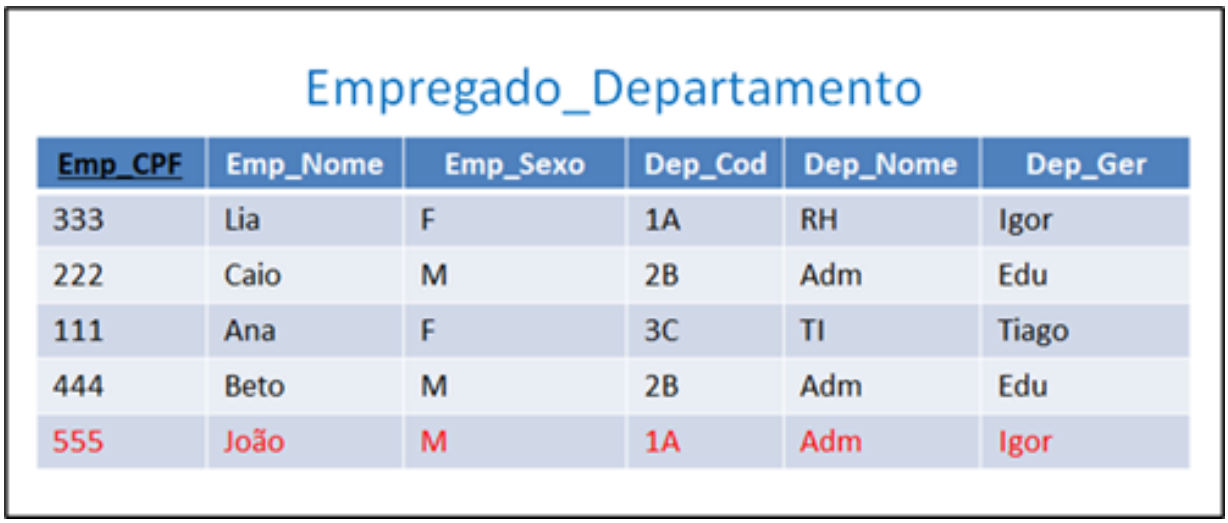
Empregado_Departamento

Emp_CPF	Emp_Nome	Emp_Sexo	Dep_Cod	Dep_Nome	Dep_Ger
333	Lia	F	1A	RH	Igor
222	Caio	M	2B	Adm	Edu
111	Ana	F	3C	TI	Tiago
444	Beto	M	2B	Adm	Edu

Figura 4.34: Tabela Empregado_Departamento.

A tabela `Empregado_Departamento`, apresentada na figura 4.34, possui uma chave primária simples, a coluna `Emp_CPF`, que representa o CPF dos empregados. Essa chave primária determina os campos `Emp_Nome`, `Emp_Sexo` e `Dep_Cod`, porém os campos `Dep_Nome` e `Dep_Ger` são determinados pelo campo `Dep_Cod` que não é a chave primária, causando a dependência transitiva, o que impede que a tabela atinja a 3FN.

A ausência da 3FN pode causar anomalias de inserção, exclusão e atualização dos registros da tabela, conforme apresentado nas figuras 4.35, 4.63 e 4.64.



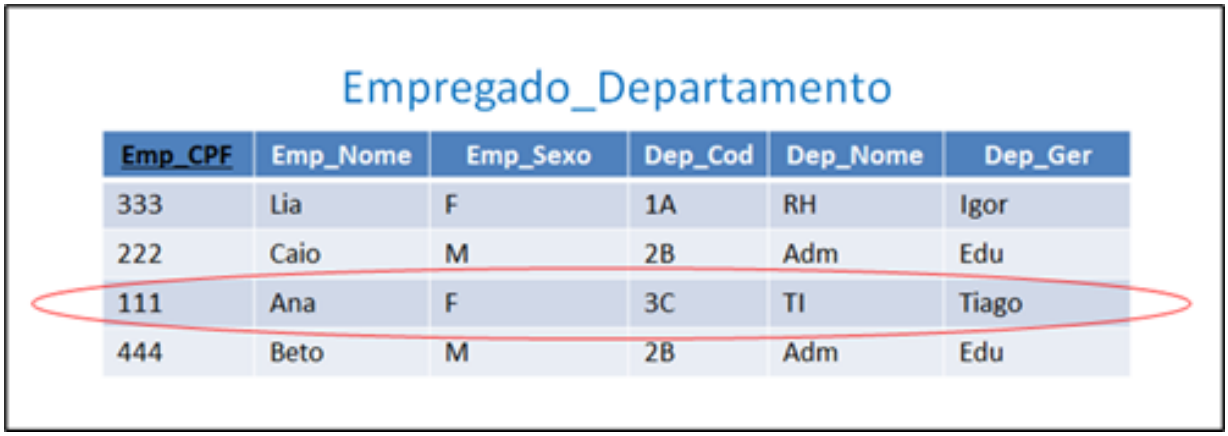
Empregado_Departamento

Emp_CPF	Emp_Nome	Emp_Sexo	Dep_Cod	Dep_Nome	Dep_Ger
333	Lia	F	1A	RH	Igor
222	Caio	M	2B	Adm	Edu
111	Ana	F	3C	TI	Tiago
444	Beto	M	2B	Adm	Edu
555	João	M	1A	Adm	Igor

Figura 4.35: Anomalia de inserção na tabela Empregado_Departamento.

De acordo com a figura 4.35, para inserir um empregado na tabela `Empregado_Departamento`, é necessário inserir todos os dados do departamento em que ele trabalha. O correto seria inserir apenas os dados do empregado, independentemente do departamento.

Na figura, para inserir o empregado cujo CPF é "555", é necessário inserir todos os dados do departamento "1A". Essa anomalia de inserção pode causar um outro problema: pode gerar dois registros do mesmo departamento com dados diferentes, como o nome ou o gerente (campo `Dep_Ger`), por exemplo.

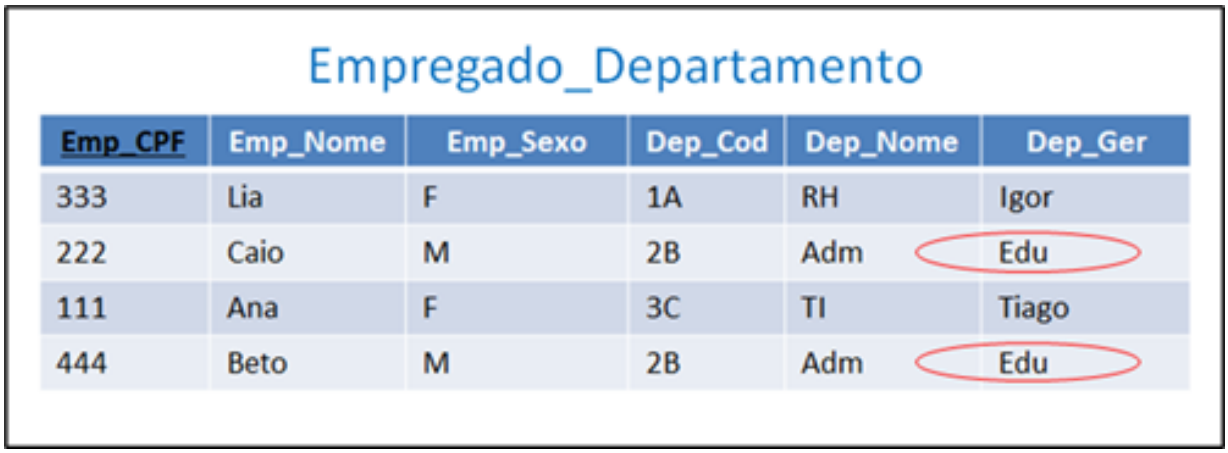


Empregado_Departamento

Emp_CPF	Emp_Nome	Emp_Sexo	Dep_Cod	Dep_Nome	Dep_Ger
333	Lia	F	1A	RH	Igor
222	Caio	M	2B	Adm	Edu
111	Ana	F	3C	TI	Tiago
444	Beto	M	2B	Adm	Edu

Figura 4.36: Anomalia de exclusão na tabela Empregado_Departamento.

De acordo com a figura 4.36, ao excluir um empregado da tabela `Empregado_Departamento`, caso esse empregado seja o único em determinado departamento, o departamento deixa de existir. Essa é uma anomalia que ocorre durante a exclusão de registros.



Empregado_Departamento

Emp_CPF	Emp_Nome	Emp_Sexo	Dep_Cod	Dep_Nome	Dep_Ger
333	Lia	F	1A	RH	Igor
222	Caio	M	2B	Adm	Edu
111	Ana	F	3C	TI	Tiago
444	Beto	M	2B	Adm	Edu

Figura 4.37: Anomalia de atualização na tabela Empregado_Departamento.

De acordo com a figura 4.37, para atualizar os dados de um departamento na tabela `Empregado_Departamento`, é necessário atualizar todos os registros dos empregados alocados a esse departamento. O correto seria atualizar os dados do departamento apenas no registro que representa esse departamento.

Na figura, para atualizar os dados do departamento "2B" é necessário atualizar os registros dos empregados com os CPFs "222" e "444", por estarem alocados a esse departamento. Essa anomalia de atualização pode causar um outro problema: pode gerar dois registros do mesmo departamento com dados diferentes, como o nome do gerente (coluna `Dep_Ger`), por exemplo, assim como ocorre na inserção.

A figura 4.38 apresenta a normalização da tabela Empregado_Departamento para a 3FN.

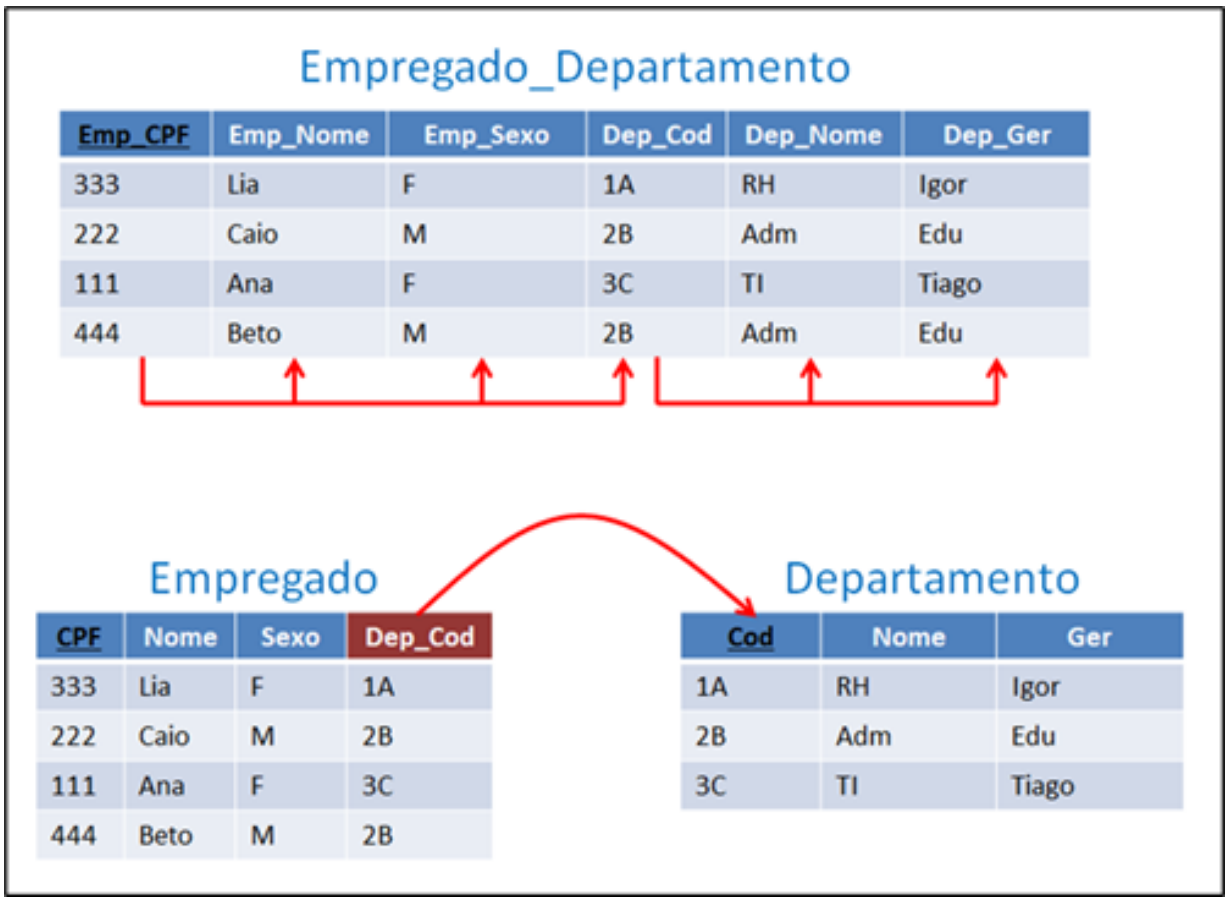


Figura 4.38: Normalização da tabela Empregado_Departamento para a 3FN.

A tabela `Empregado_Departamento` apresentada na parte superior da figura 4.38 não está na 3FN, pois os campos `Dep_Nome`, `Proj_Nome` e `Dep_Ger` apresentam dependência transitiva em relação à chave primária `Emp_CPF`.

Para atingir a 3FN, a tabela `Empregado_Departamento` foi transformada em duas tabelas: a tabela `Empregado` e a `Departamento`, conforme apresentado na parte inferior da figura. Com essa nova estrutura, é possível atingir a 3FN e eliminar as anomalias de inserção, exclusão e atualização.

Na nova estrutura, para inserir um novo empregado, basta criar um registro na tabela `Empregado`. Não é necessário inserir os dados de departamento, pois eles agora são armazenados em uma tabela à parte. Para excluir um empregado, basta deletar o registro

correspondente da tabela `Empregado` sem interferir na existência de qualquer departamento. Para atualizar os dados de um departamento, basta atualizar apenas um registro da tabela `Departamento`. Finalmente, para excluir ou mudar um empregado de departamento, basta tornar nulo ou alterar o valor do campo que é a chave estrangeira `Dep_Cod`, respectivamente.

4.4 Exemplo de projeto lógico

O projeto lógico tem por objetivo mapear o modelo conceitual, que é independente de paradigma, para um modelo lógico que contemple um paradigma específico. Aqui criaremos o modelo lógico embasado no paradigma relacional.

Esse modelo lógico será o mapeamento a partir do modelo conceitual criado na seção *3.3 Exemplo de Projeto Conceitual*.

Solução:

- **Toda pessoa cadastrada no sistema deve possuir CPF e nome e, quando possível, deve-se cadastrar o sexo também, cujos valores podem ser 'M', para masculino, e 'F', para feminino.**
- **Os pilotos devem cadastrar sua licença de voo e os passageiros, a sua categoria.**

Os requisitos apresentados anteriormente foram mapeados para o modelo conceitual apresentado na parte superior da figura 4.39. A parte inferior da mesma figura apresenta o mapeamento do modelo conceitual para o lógico.

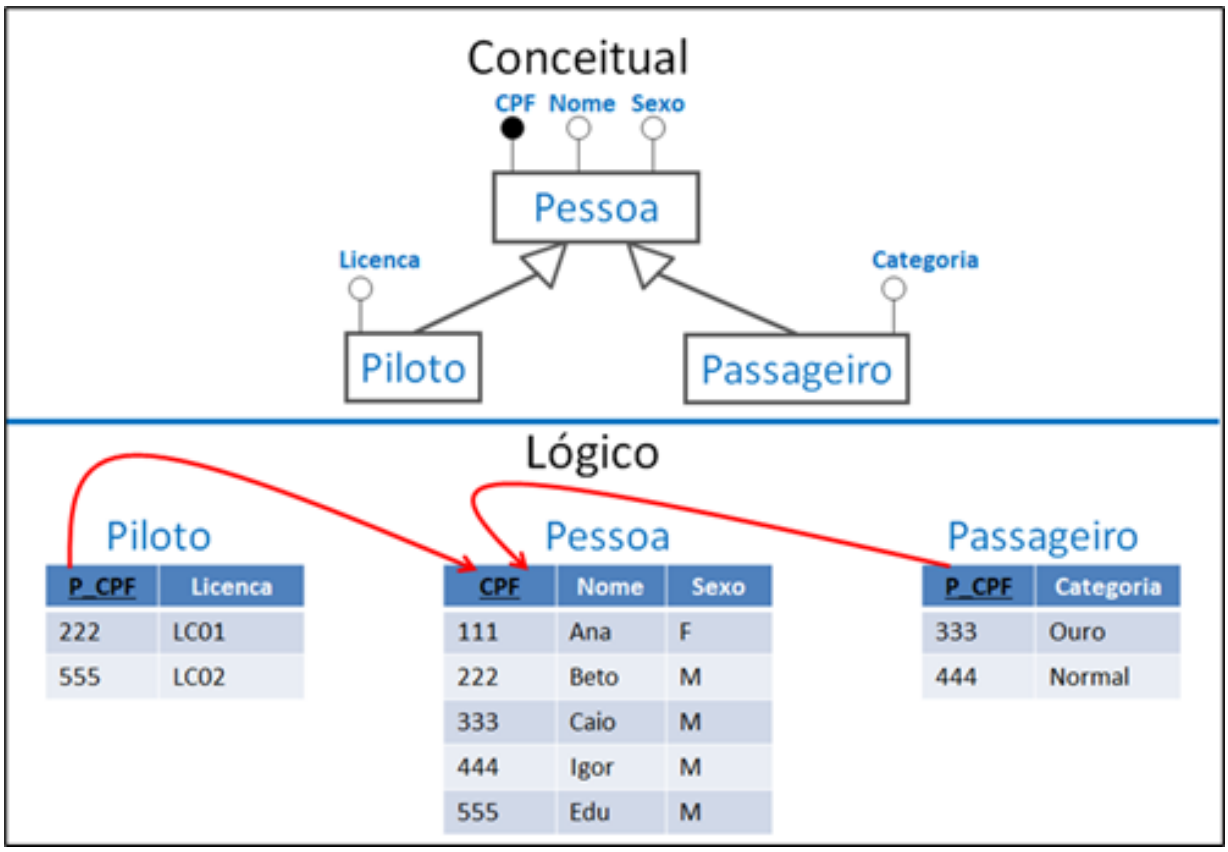


Figura 4.39: Mapeamento do relacionamento de herança do nível conceitual para o lógico.

Como pode ser observado na figura 4.39, optou-se por mapear o relacionamento de herança composto por três entidades no nível conceitual para três tabelas no nível lógico. Cabe ressaltar que existem outras soluções possíveis, conforme apresentado anteriormente.

Cada entidade foi transformada em uma tabela, os atributos das entidades foram mapeados para colunas das tabelas e o atributo identificador de cada entidade se transformou em chave primária de cada tabela. Devido ao relacionamento de herança, as chaves primárias das tabelas `Piloto` e `Passageiro` também são chaves estrangeiras e referenciam o campo `CPF` da tabela `Pessoa`.

A especificação do BDR do modelo lógico da figura 4.39 é:

- Pessoa(CPF, Nome, Sexo)
- Piloto(P_CPF, Licenca)
- P_CPF referencia Pessoa(CPF)
- Passageiro(P_CPF, Categoria)
- P_CPF referencia Pessoa(CPF)

- **Deve-se cadastrar também os telefones dos passageiros.**

O requisito apresentado anteriormente foi mapeado para o modelo conceitual apresentado na parte superior da figura 4.40. A parte inferior desta mesma figura apresenta o mapeamento do modelo conceitual para o lógico.

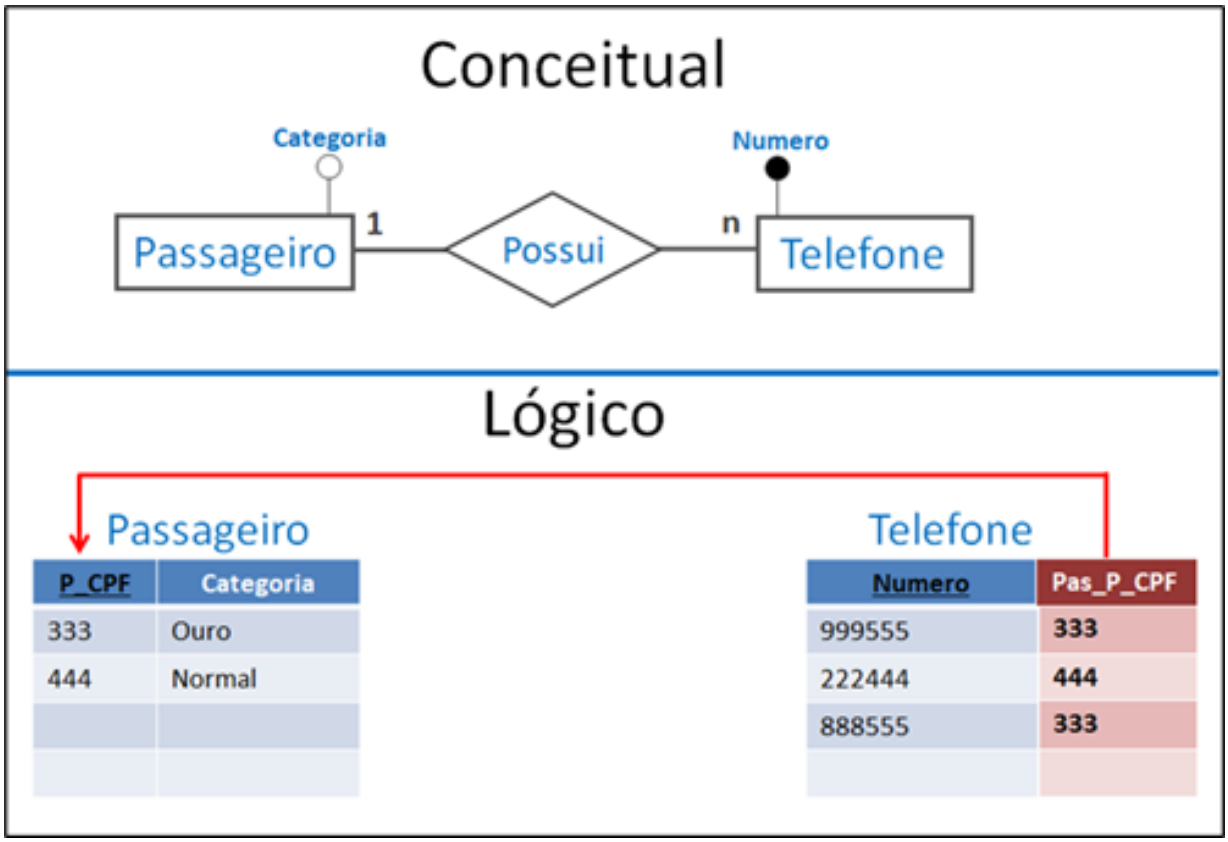


Figura 4.40: Mapeamento do relacionamento de cardinalidade 1:n entre as entidades Passageiro e Telefone do nível conceitual para o lógico.

Como pode ser observado na figura 4.40, o relacionamento `Possui`, de cardinalidade 1:n entre as entidades `Passageiro` e `Telefone`, foi mapeado para a coluna chave estrangeira `Pas_P_CPF` da tabela

`Telefone`. Essa coluna referencia a coluna `P_CPF` da tabela `Passageiro`.

A especificação da tabela `Telefone` do modelo lógico da figura 4.40 é:

- Telefone(Numero, Pas_P_CPF)
- Pas_P_CPF referencia Passageiro(P_CPF)

- **Deve-se cadastrar os dados de voo (número e data) e o respectivo piloto. Um voo é pilotado por apenas um piloto, porém um piloto pode pilotar vários voos.**

O requisito apresentado anteriormente foi mapeado para o modelo conceitual apresentado no lado esquerdo da figura 4.41. O lado direito desta mesma figura apresenta o mapeamento do modelo conceitual para o lógico.

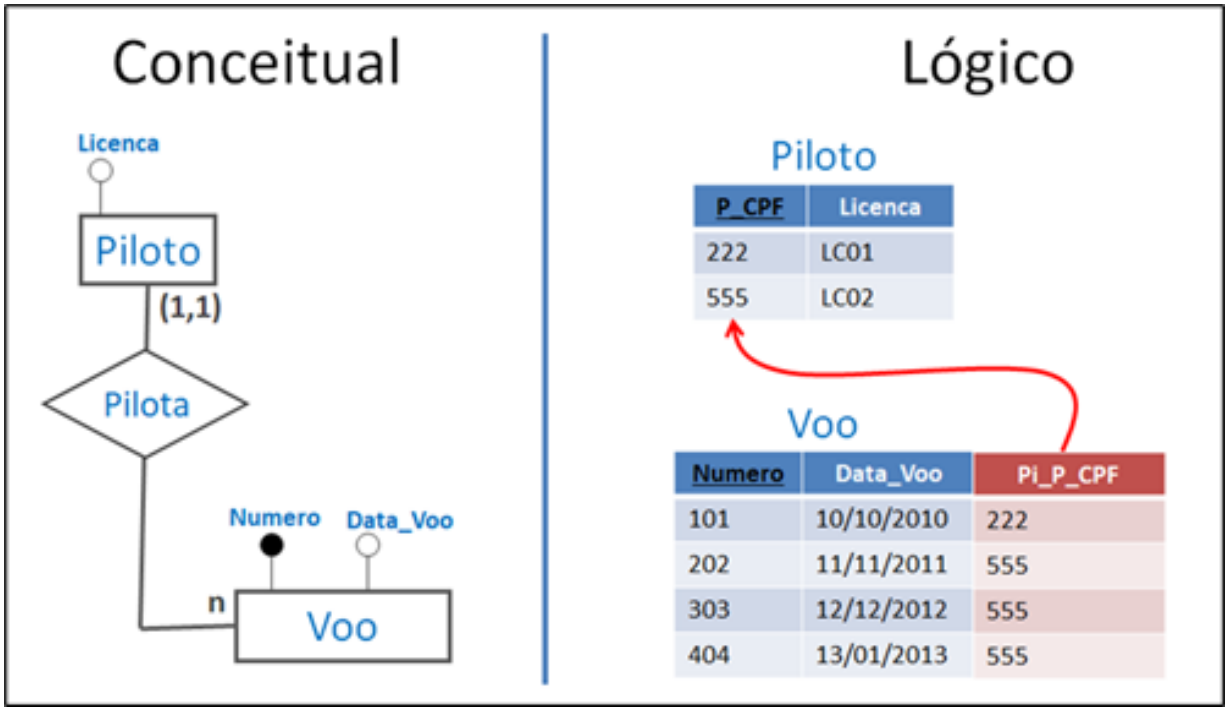


Figura 4.41: Mapeamento do relacionamento de cardinalidade 1:n entre as entidades Piloto e Voo do nível conceitual para o lógico.

Como pode ser observado na figura 4.41, o relacionamento `Pilota`, de cardinalidade 1:n entre as entidades `Piloto` e `Voo`, foi mapeado para a coluna chave estrangeira `Pi_P_CPF` da tabela `Voo`. Essa coluna referencia a coluna `P_CPF` da tabela `Piloto`.

A especificação da tabela `Voo` do modelo lógico da figura 4.41 é:

- Voo(Numero, Data_Voo, Pi_P_CPF)
- Pi_P_CPF referencia Piloto(P_CPF)

- **Deve-se identificar também passageiros dos voos.**

O requisito apresentado anteriormente foi mapeado para o modelo conceitual apresentado no lado esquerdo da figura 4.42. O lado direito desta mesma figura apresenta o mapeamento do modelo conceitual para o lógico.

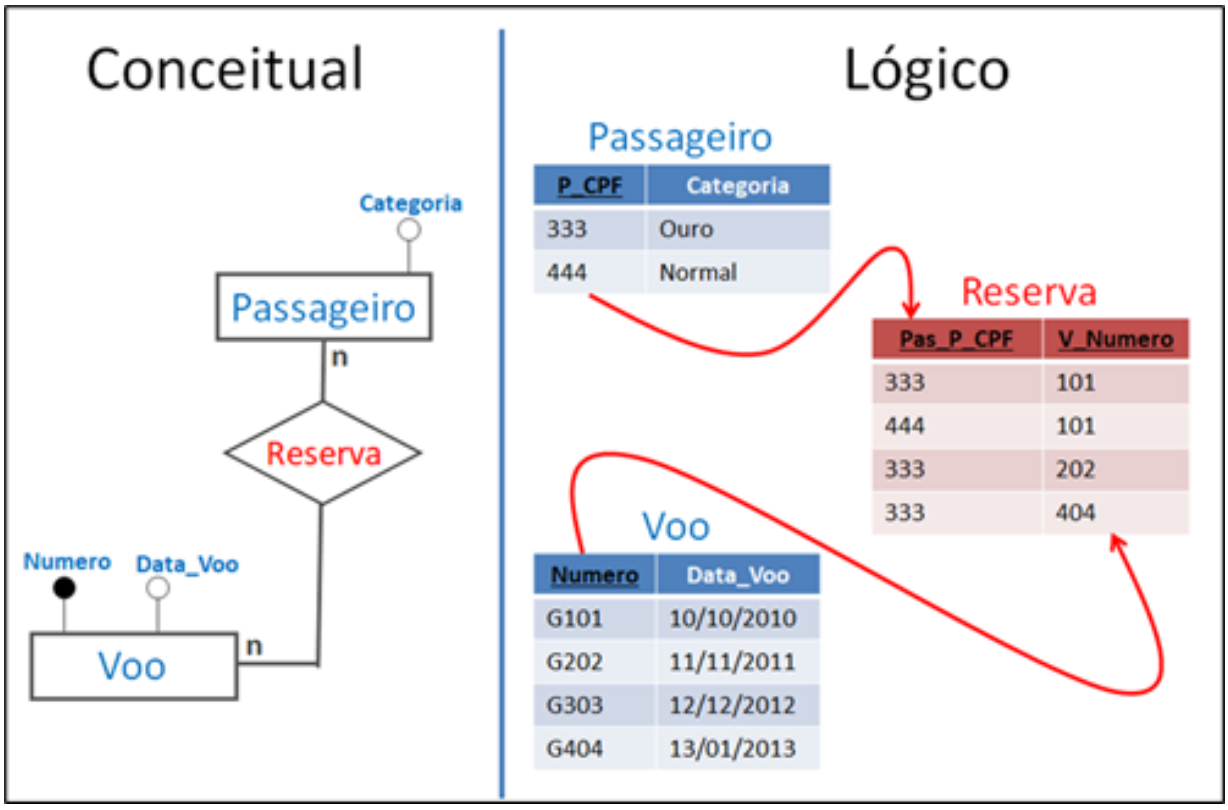


Figura 4.42: Mapeamento do relacionamento de cardinalidade n:n entre as entidades Passageiro e Voo do nível conceitual para o lógico.

Como pode ser observado na figura 4.42, o relacionamento `Reserva`, de cardinalidade n:n entre as entidades `Passageiro` e `Voo`, foi mapeado para a tabela `Reserva`. Essa tabela possui as colunas `Pas_P_CPF` e `V_Numero`, que juntas formam a chave primária. Essas duas colunas também são chaves estrangeiras. A coluna `Pas_P_CPF` referencia a coluna `P_CPF` da tabela `Passageiro` e a coluna `V_Numero` referencia a coluna `Numero` da tabela `Voo`.

A especificação da tabela `Reserva` do modelo lógico da figura 4.42 é:

- Reserva(Pas_P_CPF, V_Numero)
- Pas_P_CPF Referencia Passageiro(P_CPF)
- V_Numero Referencia Voo(Numero)

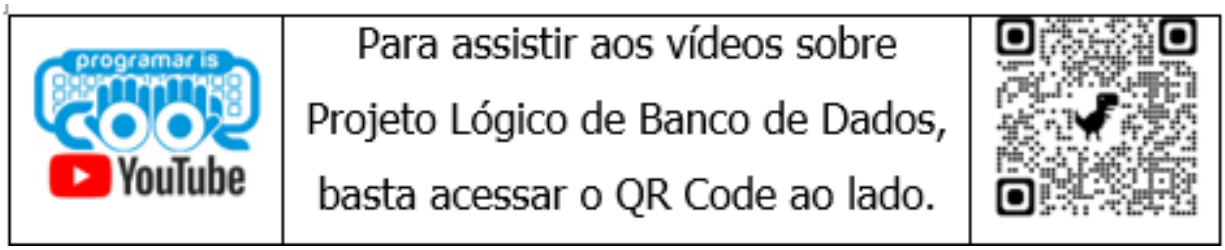


Figura 4.43: Vídeos sobre Projeto Lógico de Banco de Dados. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIDnYggoU8wBZZptjAfwZ8VC

CAPÍTULO 5
Projeto físico de banco de dados

O projeto físico do banco de dados consiste no mapeamento do projeto lógico para um SGBD específico. Entre outras atividades, deve-se definir a estrutura do banco de dados (tabelas, índices, restrições de integridade etc.), localização física de arquivos, espaço em disco a ser ocupado pelos dados do banco de dados e as permissões de acesso (grupos, usuários e papéis).

5.1 Artefatos resultantes do projeto físico

Um dos principais artefatos resultantes do projeto físico são os *scripts* (códigos) utilizados para a criação dos objetos de banco de dados no SGBD, como tabelas, índices e restrições de integridade (*constraints*), por exemplo. Esses scripts são gerados na linguagem *Structured Query Language* (SQL), comum a todos os bancos de dados relacionais.

A linguagem SQL surgiu em meados da década de 70 como resultado de um trabalho do pesquisador da IBM Edgar Frank Codd. Esse trabalho tinha como foco o desenvolvimento de uma linguagem de programação baseada no modelo de dados relacional (SILBERSCHATZ et al., 2006).

A IBM, juntamente a outros fornecedores de bancos de dados relacionais, tinha o interesse em estabelecer uma forma padronizada para acessar e manipular dados nesses bancos de dados. Por isso, com o passar do tempo, a IBM entregou a especificação da linguagem SQL para a American National Standards Institute (ANSI), um órgão normatizador reconhecido mundialmente, que lançou especificações padrões da linguagem em 1986, 1989, 1992, 1999,

2003, 2008 e 2016. Assim, a linguagem SQL tornou-se uma linguagem dinâmica, atualizada frequentemente para atender às novas demandas peculiares a todos os bancos de dados relacionais.

Os comandos da linguagem SQL podem ser organizados em várias categorias. As duas principais são:

- **Linguagem de Definição de Dados** (LDD, ou DDL, do inglês *Data Definition Language*), que abrange os comandos utilizados para criação dos objetos de banco de dados;
- **Linguagem de Manipulação de Dados** (LMD, ou DML, do inglês *Data Manipulation Language*), que abrange os comandos relacionados à inserção (`INSERT`), recuperação (`SELECT`), atualização (`UPDATE`) e exclusão (`DELETE`) dos dados armazenados em um banco de dados.

Nas próximas seções, serão apresentados os comandos LDD e LMD mais usuais, pois essa abordagem é suficiente para a alfabetização de dados, objetivo deste livro.

5.2 SQL — Linguagem de Definição de Dados (LDD)

Os comandos SQL pertencentes à categoria LDD que permitem a definição da estrutura (esquema) do banco de dados, a sua alteração e a sua exclusão são: `CREATE` , `ALTER` , `DROP` .

Comando CREATE

O comando `CREATE` pode ser utilizado para a criação de vários objetos de banco de dados, porém abordaremos apenas a criação de tabelas e restrições de integridade.

- **Sintaxe SQL para criar uma tabela:**

```
CREATE TABLE NOME_TABELA(

NOME_COLUNA-01    TIPO_COLUNA,

NOME_COLUNA-02    TIPO_COLUNA

...

);
```

- **Exemplo de comandos SQL para a criação de uma tabela:**

```
CREATE TABLE Aluno(

CPF    INT(11),

Nome     VARCHAR(30)

);
```

A figura 5.1, mais adiante, apresenta a tabela `Aluno` criada pelo comando anterior. Pode-se observar que esse comando criou a tabela `Aluno` com as colunas `CPF` e `Nome`. A coluna `CPF` pode armazenar inteiros com até 11 dígitos e a coluna `Nome` pode armazenar até 30 caracteres. Cabe ressaltar que, apesar de o exemplo ter apresentado uma tabela com apenas duas colunas, uma tabela pode possuir várias colunas.

Os comandos utilizados para a criação de tabelas permitem especificar vários tipos de restrições de integridade:

- Chave Primária: `PRIMARY KEY`;
- Chave Estrangeira: `FOREIGN KEY`;
- Restrição de Unicidade: `UNIQUE`;
- Restrição de Domínio: `CHECK`;
- Restrição de Vazio: `NOT NULL`.

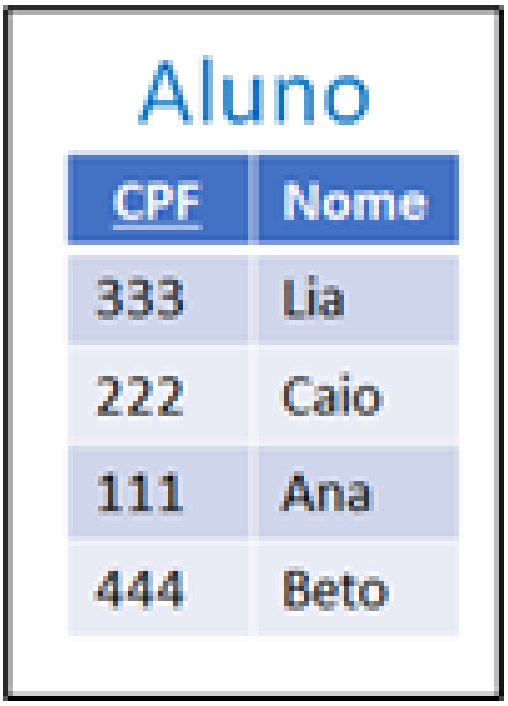


Figura 5.1: Tabela Aluno.

- **Exemplo de comandos SQL para a criação de uma tabela com restrições de integridade:**

```
CREATE TABLE Pessoa(

CPF    VARCHAR(11) NOT NULL,

Nome VARCHAR(30) NOT NULL UNIQUE,

Sexo     CHAR(1) CHECK (Sexo  IN('M', 'F')),

Emp_CNPJ VARCHAR(20),

PRIMARY KEY (CPF)

FOREIGN KEY (EMP_CNPJ)

REFERENCES  Empresa (CNPJ));
```

A figura 5.2 apresenta a tabela `Pessoa` criada pelo comando anterior. Pode-se observar que esse comando criou a tabela `Pessoa` com as colunas `CPF`, `Nome`, `Sexo` e `Emp_CNPJ`.

A coluna `CPF` pode armazenar inteiros com até 11 dígitos e é obrigatória, devido à restrição `NOT NULL`, ou seja, todo registro deverá ter um valor para o atributo `CPF`. A coluna `Nome` pode armazenar até 30 caracteres, é também é obrigatória devido à restrição `NOT NULL`, não podendo haver dois registros com o mesmo conteúdo para o atributo `Nome` nessa tabela devido à restrição `UNIQUE`. A coluna `Sexo` pode armazenar exatamente um caractere, mas ela é opcional, pois não possui a restrição `NOT NULL`; quando tiver um valor, poderá conter apenas o caractere 'M' ou 'F', devido à restrição de domínio `CHECK`.

Finalmente, a coluna `Emp_CNPJ` pode armazenar até 20 caracteres e é opcional, pois não possui a restrição `NOT NULL`; quando tiver um valor, só poderá conter um `CNPJ` contido na tabela `Empresa`, pois esse atributo é uma chave estrangeira, devido à restrição `FOREIGN KEY`, que referencia a coluna `CNPJ` da tabela `Empresa` devido ao comando `REFERENCES Empresa (CNPJ)`.

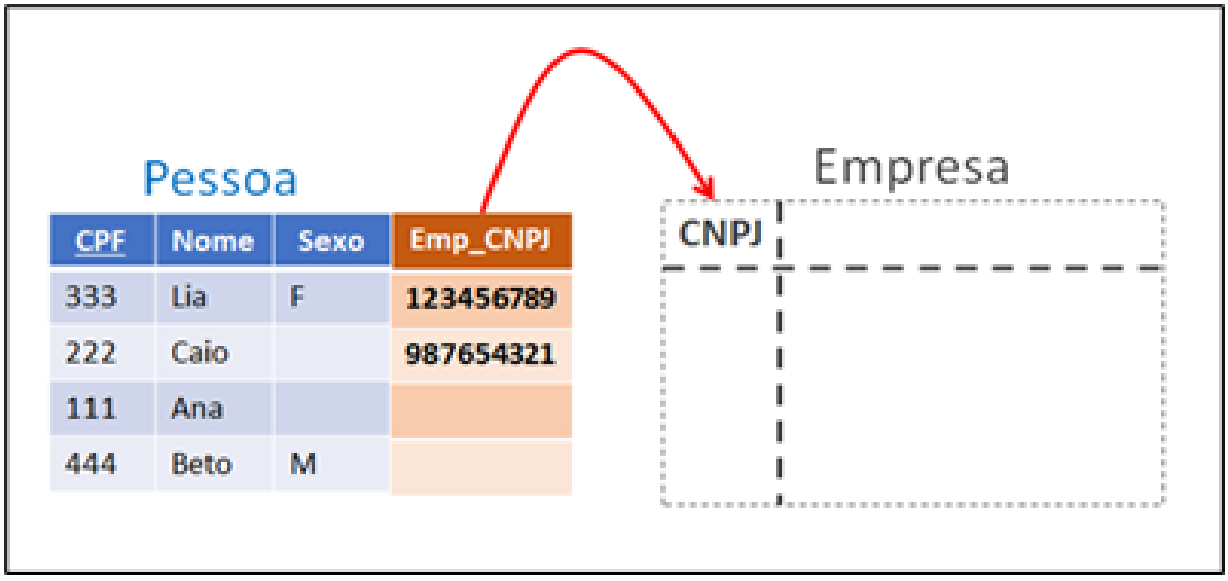


Figura 5.2: Tabela Pessoa.

Comando DROP

O comando `DROP` pode ser utilizado para a exclusão de objetos de banco de dados.

Exemplo do comando SQL para a exclusão da tabela Aluno de um banco de dados:

```
DROP TABLE Aluno;
```

Esse comando SQL exclui a tabela `Aluno` do banco de dados.

Comando ALTER

O comando `ALTER` pode ser utilizado em várias situações, dentre elas:

- Incluir novas colunas em uma tabela;
- Adicionar a definição de uma restrição de integridade em uma tabela;
- Excluir colunas existentes em uma tabela;
- Excluir uma restrição de integridade existente em uma tabela;
- Modificar uma coluna.

Exemplo dos comandos SQL para a alteração de uma tabela para acrescentar uma coluna nova e uma restrição de integridade referencial após a sua criação:

```
CREATE TABLE Pessoa(

CPF    VARCHAR(11) NOT NULL,

Nome VARCHAR(30) NOT NULL UNIQUE,

Sexo CHAR(1) CHECK (Sexo  IN('M', 'F'))

);

ALTER TABLE Pessoa  ADD COLUMN Emp_CNPJ VARCHAR(20);
```

```
ALTER TABLE Pessoa ADD CONSTRAINT FK_Empresa FOREIGN KEY (EMP_CNPJ)

REFERENCES  Empresa (CNPJ);
```

Os comandos SQL apresentados anteriormente foram utilizados para criar a mesma tabela `Pessoa` apresentada na figura 5.2. Primeiro, a tabela `Pessoa` foi criada com as colunas `CPF`, `Nome` e `Sexo`. Após esse passo, foi acrescentada a coluna `Emp_CNPJ` por meio dos comandos `ALTER TABLE` e `ADD COLUMN`. Após acrescentar essa coluna, foi criada a restrição de integridade referencial por meio dos comandos `ALTER TABLE`, `ADD`, `CONSTRAINT`, `FOREIGN KEY` e `REFERENCES`.

- **Exemplo dos comandos SQL para a alteração de uma tabela para excluir uma coluna e uma restrição de integridade.**

```
ALTER TABLE Pessoa DROP FOREIGN KEY ‘FK_Empresa’;

ALTER TABLE Pessoa  DROP COLUMN Emp_CNPJ;
```

Os comandos SQL apresentados anteriormente foram utilizados para excluir a restrição de integridade referencial denominada `FK_Empresa` da tabela `Pessoa` por meio dos comandos `ALTER TABLE` e `DROP FOREIGN KEY` e, também, para excluir a coluna `Emp_CNPJ` da mesma tabela por meio dos comandos `ALTER TABLE` e `DROP COLUMN`.

A seguir, será apresentada a criação do script na linguagem SQL (comandos LDD) para a geração das tabelas e restrições de integridade do banco de dados resultante do mapeamento do modelo lógico apresentado na seção *4.4 Exemplo de Projeto Lógico* para o modelo físico. Cabe ressaltar que esse procedimento é apenas uma parte do projeto físico.

5.3 Exemplo de Script de Geração do Banco de Dados

Neste exemplo, apresentaremos os scripts na linguagem SQL (comandos LDD) de criação das tabelas e restrições de integridade do modelo lógico apresentado na seção *4.4 Exemplo de projeto lógico*. O objetivo é apresentar apenas uma parte do mapeamento do projeto lógico para o projeto físico, pois o projeto físico possui outros artefatos que fogem ao escopo deste livro.

Solução:

- **Toda pessoa cadastrada no sistema deve possuir CPF e nome, e, quando possível, deve-se cadastrar o sexo também, cujos valores podem ser apenas 'M', para masculino, e 'F', para feminino.**
- **Os pilotos devem cadastrar sua licença de voo e os passageiros, a sua categoria.**

Os requisitos apresentados anteriormente foram mapeados para as tabelas `Pessoa`, `Piloto` e `Passageiro`, apresentadas na figura 5.3.

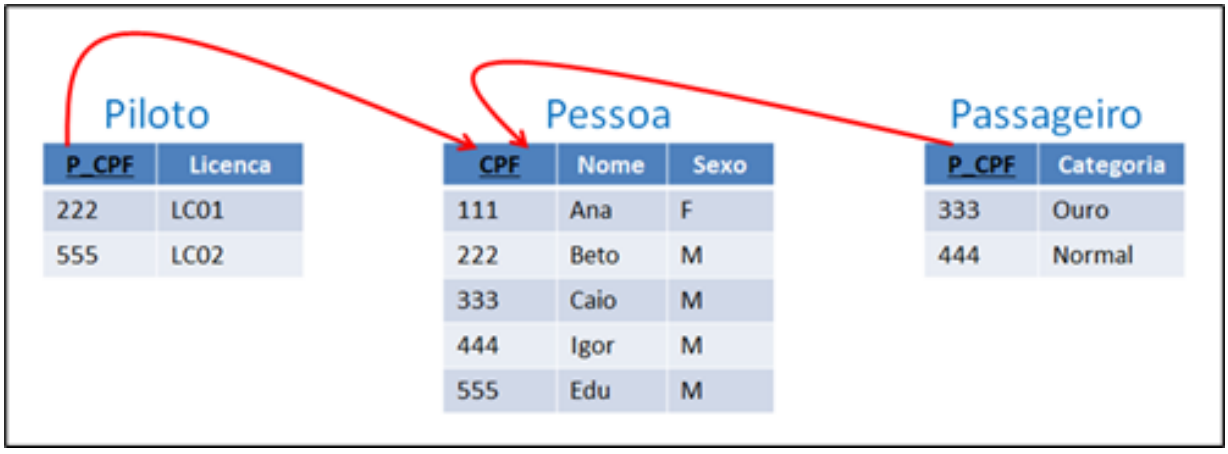


Figura 5.3: Tabelas Pessoa, Piloto e Passageiro.

O script SQL para a criação da tabela `Pessoa` apresentada na figura 5.3 é o seguinte:

```
CREATE TABLE Pessoa(

CPF    VARCHAR(11) NOT NULL,

Nome     VARCHAR(30)       NOT NULL    UNIQUE,

Sexo     CHAR(1)          CHECK (Sexo  IN('M, 'F')),

PRIMARY KEY (CPF));
```

Como pode ser observado no script anterior, a tabela `Pessoa` foi criada com as colunas `CPF`, `Nome` e `Sexo`. Os atributos `CPF` e `Nome` são obrigatórios, sendo que o atributo `CPF` é a chave primária da tabela e o atributo `Nome` não pode possuir o mesmo valor em dois registros diferentes, ou seja, ele é único. O atributo `Sexo` é opcional, mas quando possuir um valor, obrigatoriamente poderá conter apenas os caracteres 'M' ou 'F'.

O script SQL para a criação da tabela `Piloto` apresentada na figura 5.3 é o seguinte:

```
CREATE TABLE Piloto(

P_CPF            VARCHAR(11)   NOT NULL,

Licenca       VARCHAR(5)     NOT NULL UNIQUE,

PRIMARY KEY (P_CPF),

FOREIGN KEY (P_CPF)

REFERENCES  Pessoa (CPF));
```

Como pode ser observado no script anterior, a tabela `Piloto` foi criada com as colunas `P_CPF` e `Licenca`. Ambas são obrigatórias, sendo que o atributo `P_CPF` é a chave primária e estrangeira da tabela. Esse atributo referencia o atributo `CPF` da tabela `Pessoa`. Para criar um registro na tabela `Piloto`, é necessário que o valor do atributo `P_CPF` exista no campo `CPF` de um registro da tabela

`Pessoa`, devido à restrição de integridade referencial. Ainda na tabela `Piloto`, o atributo `Licenca` não pode possuir o mesmo valor em dois registros diferentes, ou seja, ele é único.

O script SQL para a criação da tabela `Passageiro` apresentada na figura 5.3 é o seguinte:

```
CREATE TABLE Passageiro(

P_CPF      VARCHAR(11)          NOT NULL,

Categoria  VARCHAR(10) ,

PRIMARY KEY (P_CPF),

FOREIGN KEY (P_CPF)

REFERENCES  Pessoa (CPF));
```

Como pode ser observado no script anterior, a tabela `Passageiro` foi criada com as colunas `P_CPF` e `Categoria`. O atributo `P_CPF` é obrigatório, e é a chave primária e estrangeira da tabela. Esse atributo referencia o atributo `CPF` da tabela `Pessoa`. Para criar um registro na tabela `Passageiro`, é necessário que o valor do atributo `P_CPF` exista no campo `CPF` de um registro da tabela `Pessoa`, devido à restrição de integridade referencial.

- **Deve-se cadastrar também os telefones dos passageiros.**

O requisito apresentado anteriormente foi mapeado para a tabela `Telefone`, apresentada na figura 5.4.

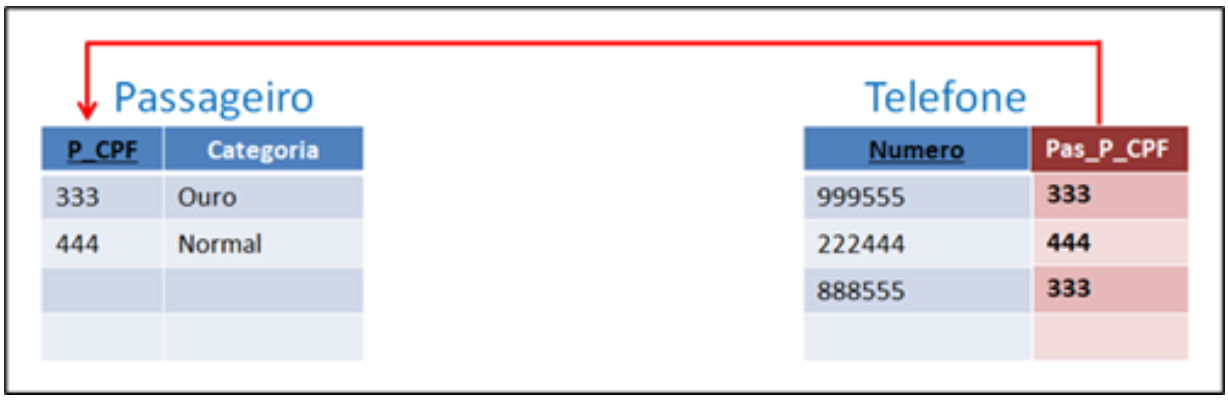


Figura 5.4: Tabela Telefone.

O script SQL para a criação da tabela `Telefone` apresentada na figura 5.4 é o seguinte:

```
CREATE TABLE Telefone(

Numero          VARCHAR(10)     NOT NULL,

Pas_P_CPF           VARCHAR(11)     NOT NULL,

PRIMARY KEY (Numero),

FOREIGN KEY (Pas_P_CPF)

REFERENCES  Passageiro(P_CPF));
```

Como pode ser observado no script anterior, a tabela `Telefone` foi criada com as colunas `Numero` e `Pas_P_CPF`, e ambas são obrigatórias. O atributo `Numero` é a chave primária da tabela e o atributo `Pas_P_CPF` é a chave estrangeira que referencia o atributo `P_CPF` da tabela `Passageiro`.

- **Deve-se cadastrar os dados de voo, número e data, e o respectivo piloto. Um voo é pilotado por apenas um piloto, porém um piloto pode pilotar vários voos.**

Os requisitos apresentados anteriormente foram mapeados para a tabela `Voo` apresentada na figura 5.5.

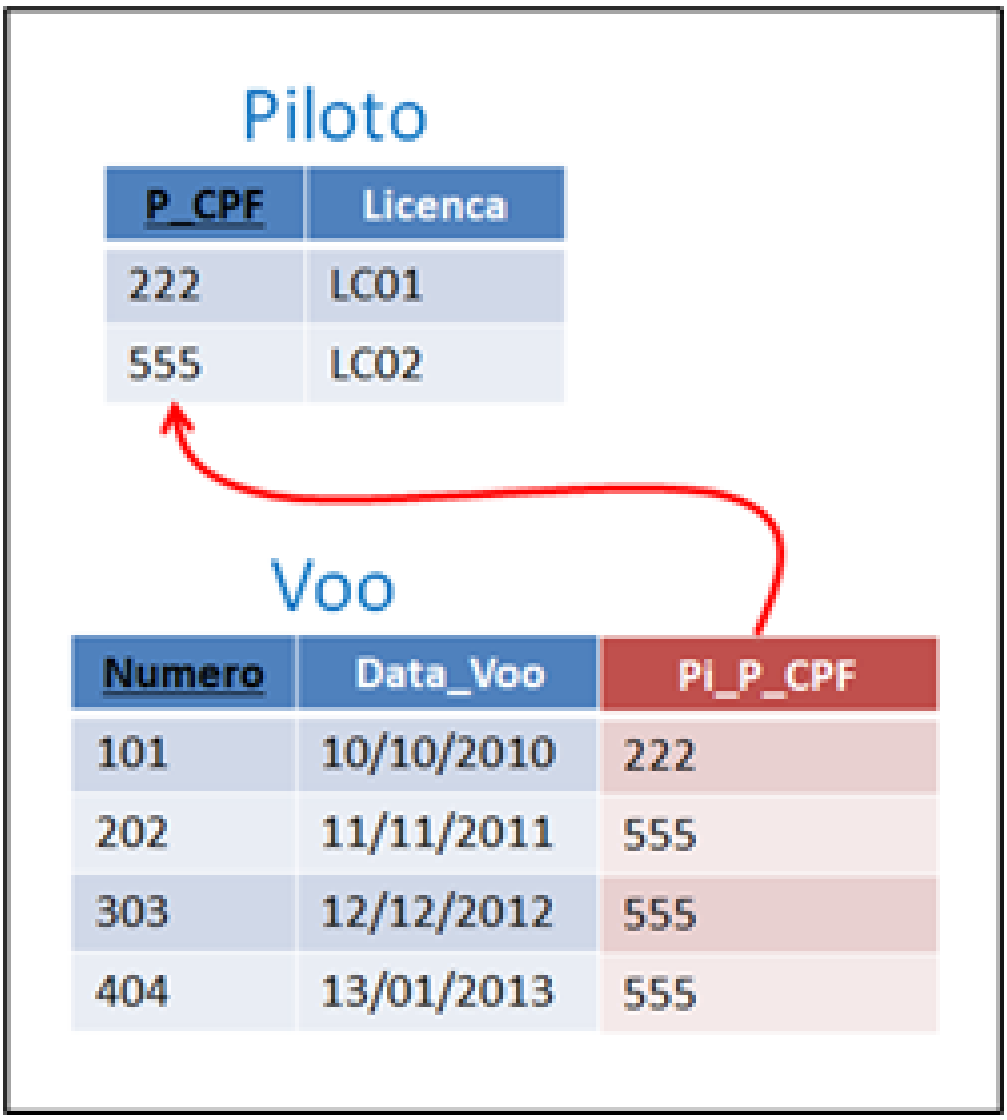


Figura 5.5: Tabela Voo.

O script SQL para a criação da tabela `Voo` apresentada na figura 5.5 é o seguinte:

```
CREATE TABLE Voo(
Numero    Integer                  NOT NULL,

Data_Voo     Date              NOT NULL,

Pi_P_CPF  VARCHAR(11)      NOT NULL,

PRIMARY KEY (Numero),
```

```
FOREIGN KEY (Pi_P_CPF)

REFERENCES  Piloto (P_CPF));
```

Como pode ser observado no script anterior, a tabela `Voo` foi criada com as colunas `Numero`, `Data_Voo` e `Pi_P_CPF`. Todos são obrigatórios: o atributo `Numero` é a chave primária da tabela e o atributo `Pi_P_CPF` é a chave estrangeira que referencia o atributo `P_CPF` da tabela `Piloto`.

- **Deve-se identificar também passageiros dos voos.**

O requisito apresentado anteriormente foi mapeado para a tabela `Reserva`, apresentada na figura 5.6.

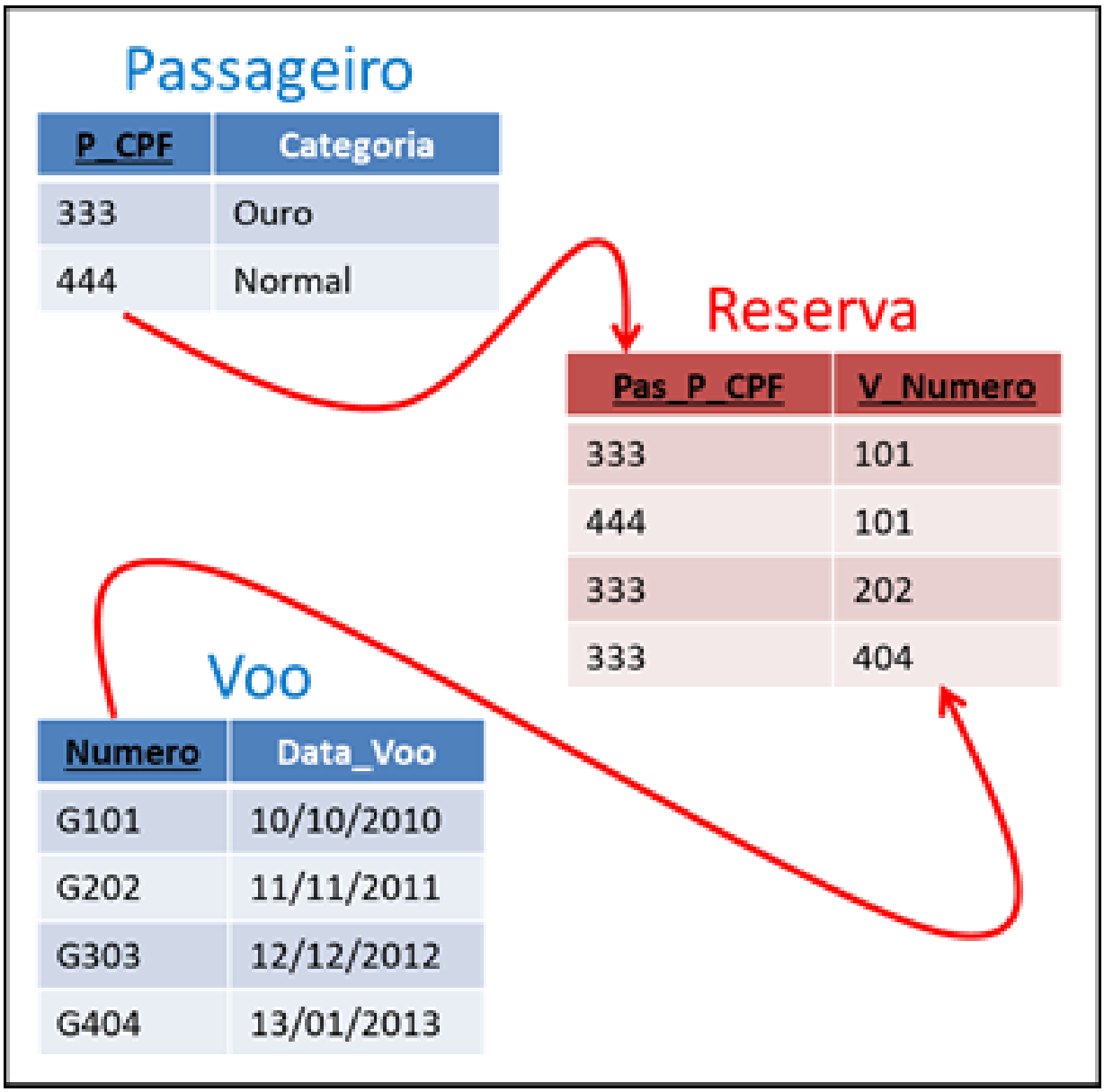


Figura 5.6: Tabela Reserva.

O script SQL para a criação da tabela `Reserva` apresentada na figura 5.6 é o seguinte:

```
CREATE TABLE Reserva(

PAS_P_CPF          VARCHAR(11)       NOT NULL,

V_Numero     Integer           NOT NULL,
```

```
PRIMARY KEY (Pas_P_CPF, V_Numero),

FOREIGN KEY (Pas_P_CPF)

REFERENCES  Passageiro(P_CPF),

FOREIGN KEY (V_Numero)

REFERENCES  Voo(Numero));
```

Como pode ser observado no script anterior, a tabela `Reserva` foi criada com as colunas `Pas_P_CPF` e `V_Numero`, e ambas são obrigatórias, pois compõem a chave primária da tabela. Cada uma dessas colunas são chaves estrangeiras: o atributo `Pas_P_CPF` referencia o atributo `Pas_P_CPF` da tabela `Passageiro` e o atributo `V_Numero` referencia o atributo `Numero` da tabela `Voo`.

5.4 SQL — Linguagem de Manipulação de Dados (DML)

Os comandos SQL pertencentes à categoria DML que permitem a inserção, a recuperação, a atualização e a exclusão de dados em tabelas do banco de dados são: `INSERT`, `SELECT`, `UPDATE` e `DELETE`.

Comando INSERT

O comando `INSERT` é utilizado para a inserção de registros nas tabelas.

- **Exemplo do uso do comando SQL para a inserção de linhas na tabela Pessoa:**

```
INSERT INTO Pessoa(CPF, Nome, Sexo)
```

```
VALUES('111', 'Ana', 'F'),

('222', 'Beto', 'M'),

('333', 'Caio', 'M'),

('444', 'Igor', 'M'),

('555', 'Edu', 'M');
```

A figura 5.7 apresenta a tabela `Pessoa` com os registros inseridos por meio do comando `INSERT` anterior. Pode-se observar que esse comando criou os registros da tabela `Pessoa` com as colunas `CPF` e `Sexo`.

CPF	Nome	Sexo
111	Ana	F
222	Beto	M
333	Caio	M
444	Igor	M
555	Edu	M

Figura 5.7: Tabela Pessoa após inserção dos registros.

Comando SELECT

O comando `SELECT` é utilizado para a recuperação de registros das tabelas.

- **Exemplo do uso dos comandos SQL para a recuperação de todas as linhas e todas as colunas da tabela Pessoa:**

```
SELECT CPF, Nome, Sexo

FROM Pessoa;

SELECT *

FROM Pessoa;

SELECT Pessoa.*

FROM Pessoa;
```

- **Exemplo do uso do comando SQL para a recuperação das colunas CPF e Nome de todas as linhas da tabela Pessoa:**

```
SELECT CPF, Nome

FROM Pessoa;
```

- **Exemplo do uso do comando SQL para a recuperação das colunas CPF e Nome dos registros da tabela Pessoa que respeitam uma determinada condição, nesse caso, pessoas do sexo masculino:**

```
SELECT CPF, Nome

FROM Pessoa

WHERE Sexo = 'M';
```

Comando UPDATE

O comando `UPDATE` é utilizado para a alteração de atributos dos registros das tabelas.

- **Exemplo do uso do comando SQL para a alteração do atributo Nome tabela Pessoa cujo CPF = 111:**

```
UPDATE Pessoa

SET Nome = ‘Maria’

WHERE CPF = ‘111’;
```

Comando DELETE

O comando `DELETE` é utilizado para exclusão de registros das tabelas.

- **Exemplo do uso do comando SQL para a exclusão do registro da tabela Pessoa cujo CPF = 111:**

```
DELETE FROM Pessoa

WHERE CPF = ‘111’;
```

- **Exemplo do uso do comando SQL para a exclusão de todos os registros da tabela Pessoa:**

```
DELETE FROM Pessoa;
```

Comandos COMMIT e ROLLBACK

Quando um usuário do banco de dados realiza inserções, alterações ou exclusões dos registros de uma tabela, apenas ele, por meio da sua conexão com o banco de dados, enxerga os efeitos das operações realizadas; os outros usuários do banco de dados só enxergarão esses efeitos quando eles se tornarem permanentes. O comando `COMMIT` torna permanente os efeitos dessas operações, assim todos os usuários os enxergarão.

O comando `ROLLBACK` permite a um usuário desfazer os efeitos das operações de inserção, alteração ou exclusão dos registros de uma tabela que ainda não foram confirmados, ou seja, não se tornaram permanentes. Assim, quando um usuário pretende desfazer os efeitos de inserções, alterações ou exclusões de registros em uma tabela, ele deve executar o comando `ROLLBACK` . Cabe ressaltar que o

comando `ROLLBACK` só reverte os efeitos das operações não confirmadas pelo comando `COMMIT` .

Nesta seção, foram apresentados os comandos DML básicos, utilizados para manipular dados de apenas uma tabela de cada vez. No entanto, o mais comum em consultas SQL é a manipulação de dados de várias tabelas simultaneamente. Para isso, utilizamos os operadores de conjuntos e operadores relacionais. Os operadores mais importantes serão apresentados nas próximas seções.

5.5 Operadores de Conjuntos

Nesta seção serão apresentados os operadores `UNION` (união), `INTERSECT` (interseção) e `EXCEPT` (diferença).

Operador UNION (U)

- Operador binário, ou seja, opera sobre duas tabelas;
- Constrói uma tabela consistindo em todas as linhas que aparecem nas tabelas específicas;
- Elimina registros duplicados;
- As duas tabelas devem possuir o mesmo número de colunas e cada par de colunas correspondentes serem do mesmo tipo de dados (compatíveis para união).

Exemplo do uso do comando UNION

```
SELECT * FROM Pessoa

UNION

SELECT * FROM Cliente
```

A figura 5.8 apresenta a tabela resultante do operador `UNION` entre as tabelas `Pessoa` e `Cliente` apresentado anteriormente.

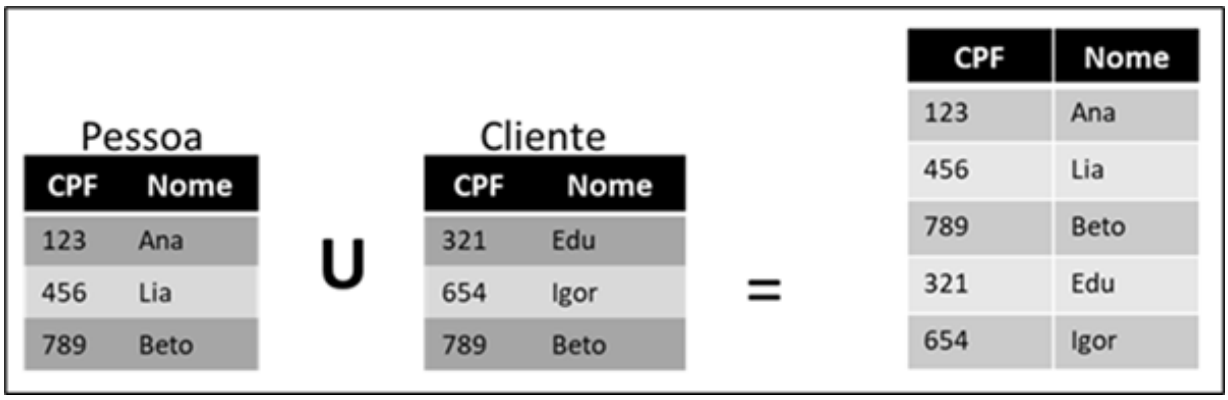


Figura 5.8: Operador UNION.

Como pode ser observado na figura 5.8, a tabela resultante da união das tabelas Pessoa e `Cliente` possui os registros existentes nessas duas tabelas. Cabe ressaltar que o registro cujo `CPF` é 789 só aparece uma única vez na tabela resultante, pois, apesar de esse registro existir nas duas tabelas, o resultado da união elimina duplicatas.

Operador INTERSECT (∩)

- Operador binário, ou seja, opera sobre duas tabelas;
- Constrói uma tabela consistindo em todas as linhas que aparecem em ambos os pares de relações específicas.

Exemplo do uso do comando INTERSECT

```
SELECT * FROM Pessoa

INTERSECT

SELECT * FROM Cliente
```

A figura 5.9 apresenta a tabela resultante do operador `INTERSECT` entre as tabelas `Pessoa` e `Cliente` apresentado anteriormente.

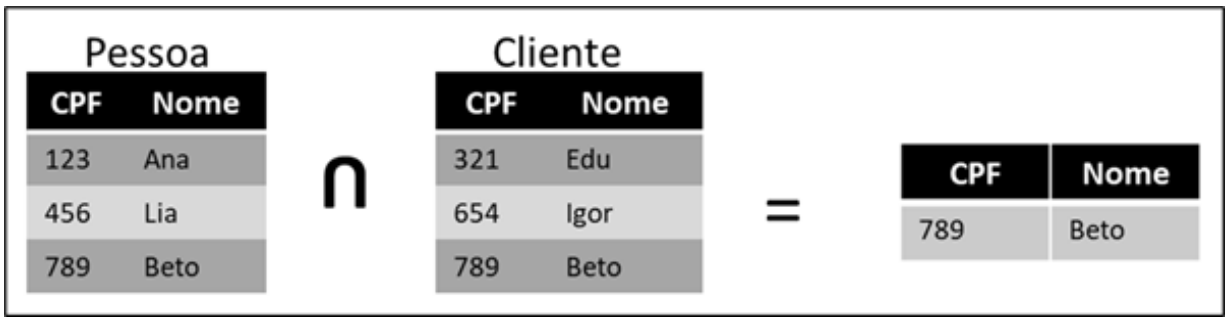


Figura 5.9: Operador INTERSECT.

Como pode ser observado na figura 5.9, a tabela resultante da interseção das tabelas `Pessoa` e `Cliente` possui apenas o registro que existe nessas duas tabelas. Cabe ressaltar que o registro cujo CPF é 789 é o único que aparece na tabela resultante, pois apenas esse registro existe nas duas tabelas.

Operador EXCEPT (-)

- Operador binário, ou seja, opera sobre duas tabelas;
- Constrói uma tabela consistindo em todas as linhas que aparecem na primeira, mas não na segunda, do par de relações específicas.

Exemplo do uso do comando EXCEPT

```
SELECT * FROM Pessoa

EXCEPT

SELECT * FROM Cliente
```

A figura 5.10 apresenta a tabela resultante do operador `EXCEPT` entre as tabelas `Pessoa` e `Cliente` apresentado anteriormente.

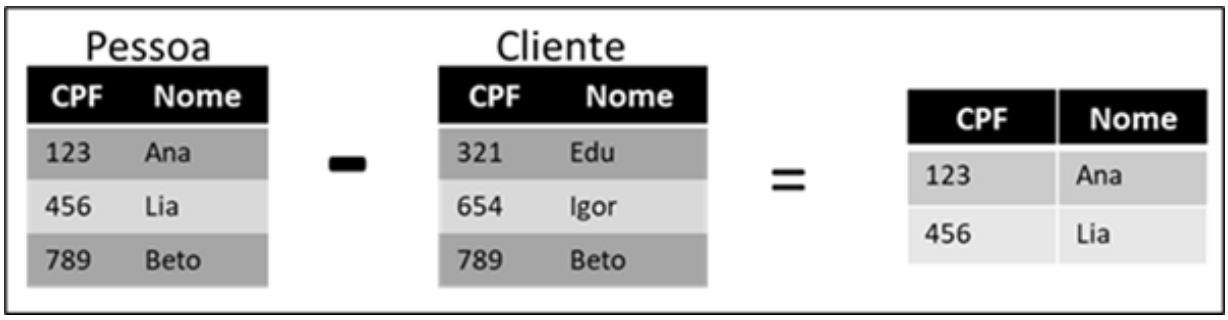


Figura 5.10: Operador EXCEPT.

Como pode ser observado na figura 5.10, a tabela resultante do uso do operador `EXCEPT` é equivalente ao complemento da tabela `Cliente` em relação à tabela `Pessoa`, ou seja, as linhas que existem na tabela `Pessoa` e não existem na tabela `Cliente`. A tabela resultante possui apenas os registros cujos CPFs são 123 e 456, pois esses registros existem na tabela `Pessoa`, mas não existem na tabela `Cliente`. Alguns bancos de dados usam o termo `MINUS` no lugar do termo `EXCEPT` para realizar essa operação.

Junção

A junção é utilizada para a recuperação de dados de várias tabelas simultaneamente. Nesta seção, serão apresentados os principais tipos de junções. A figura 5.11 apresenta as tabelas `Empregado` e `Departamento`, com os respectivos registros, que serão utilizadas para exemplificar as consultas utilizando junções.

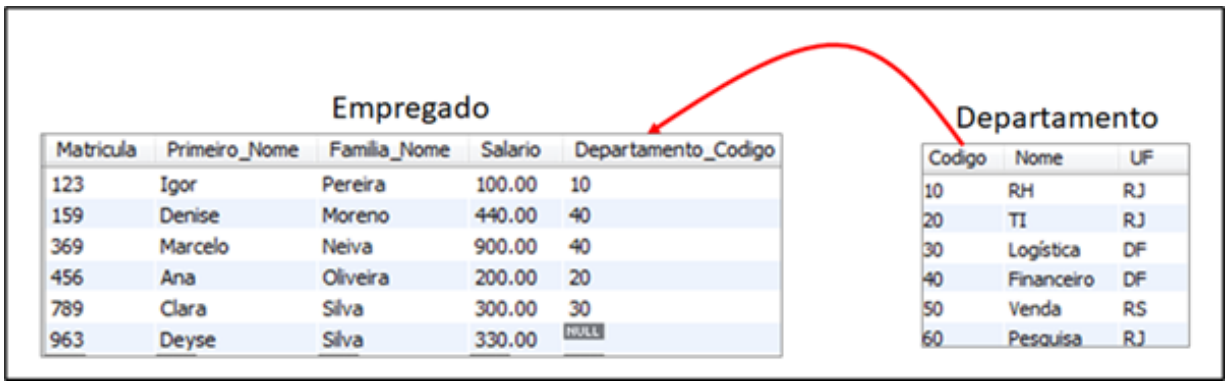


Figura 5.11: Tabelas Empregado e Departamento.

Como pode ser observado na figura 5.11, a tabela `Empregado` possui as colunas `Matricula`, `Primerio_Nome`, `Familia_Nome`, `Salario` e `Departamento_Codigo`. Por outro lado, a tabela `Departamento` possui as colunas `Codigo`, `Nome` e `UF`.

A coluna `Departamento_Codigo` em `Empregado` é uma chave estrangeira que referencia a coluna `Codigo` em `Departamento`. Observa-se também que o registro na tabela `Empregado` cuja matrícula é 963 não está associado a nenhum registro da tabela `Departamento`, e que os registros na tabela `Departamento` com os códigos 50 e 60 não estão associados a nenhum registro de `Empregado`.

Do exposto, pode-se concluir que o relacionamento entre as tabelas `Departamento` e `Empregado` é de cardinalidade 1:n, ou seja, um departamento pode possuir zero ou vários empregados, porém um empregado pode ser alocado em zero ou um departamento.

CROSS JOIN (Junção Cruzada)

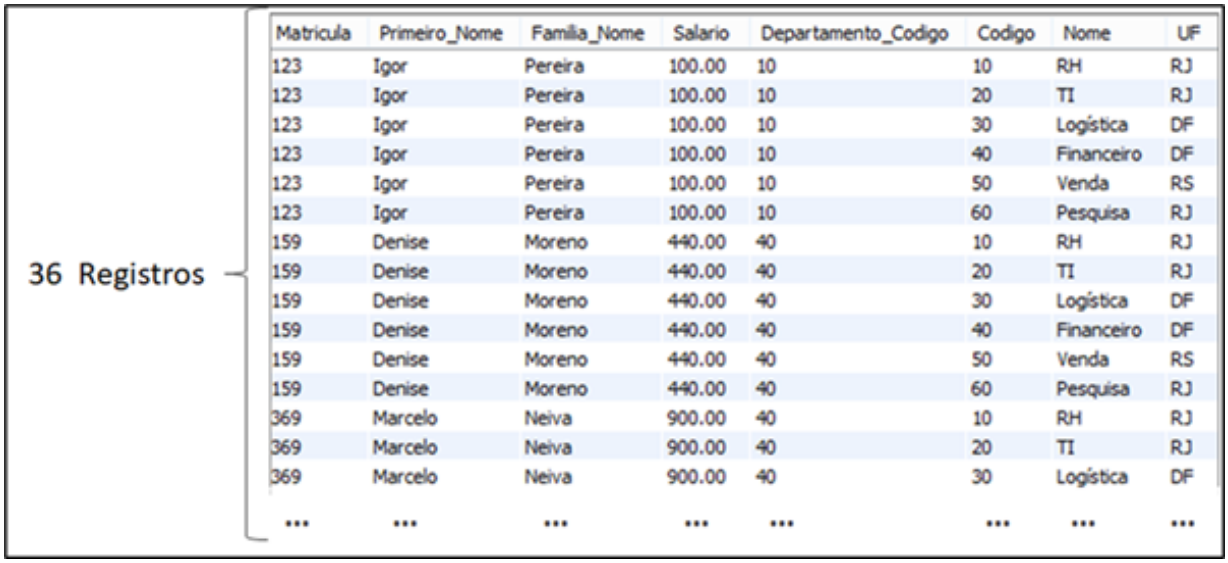
A operação de junção cruzada é utilizada para realizar a combinação de todas as linhas de duas tabelas.

Exemplo de Junção Cruzada

```
SELECT *

FROM Empregado CROSS JOIN Departamento;
```

Ao ser executada, a consulta apresentada gerará uma nova tabela composta por todos os atributos das duas tabelas com a quantidade de linhas igual ao número de linhas da tabela `Empregado`, que neste caso é 6, vezes o número de linhas da tabela `Departamento`, que neste caso também é 6, resultando em 36 linhas. A figura 5.12 mais adiante apresenta a junção cruzada entre as tabelas `Empregado` e `Departamento`.

O caractere `*` na consulta representa todas as colunas das duas tabelas. É possível recuperar apenas um subconjunto das colunas: basta substituir o caractere `*` pelos nomes das colunas a serem recuperadas.



Matricula	Primeiro_Nome	Familia_Nome	Salario	Departamento_Codigo	Codigo	Nome	UF
123	Igor	Pereira	100.00	10	10	RH	RJ
123	Igor	Pereira	100.00	10	20	TI	RJ
123	Igor	Pereira	100.00	10	30	Logística	DF
123	Igor	Pereira	100.00	10	40	Financeiro	DF
123	Igor	Pereira	100.00	10	50	Venda	RS
123	Igor	Pereira	100.00	10	60	Pesquisa	RJ
159	Denise	Moreno	440.00	40	10	RH	RJ
159	Denise	Moreno	440.00	40	20	TI	RJ
159	Denise	Moreno	440.00	40	30	Logística	DF
159	Denise	Moreno	440.00	40	40	Financeiro	DF
159	Denise	Moreno	440.00	40	50	Venda	RS
159	Denise	Moreno	440.00	40	60	Pesquisa	RJ
369	Marcelo	Neiva	900.00	40	10	RH	RJ
369	Marcelo	Neiva	900.00	40	20	TI	RJ
369	Marcelo	Neiva	900.00	40	30	Logística	DF
...	...	...	...	...	...	...	...

Figura 5.12: Resultado do CROSS JOIN entre as tabelas Empregado e Departamento.

A figura 5.12 apresenta apenas parte dos registros resultantes da junção cruzada entre as tabelas `Empregado` e `Departamento`, porém ressalta que o resultado completo são 36 registros. É possível observar que os registros da tabela `Empregado` cuja matrícula é 123 e 159 se relacionam com todos os seis registros da tabela `Departamento` cujos códigos são 10. 20, 30, 40 50 e 60. Isso acontece com todos

os registros da tabela `Empregado`, gerando os 36 registros, porém por problema de espaço a figura foi cortada.

INNER JOIN (Junção Interna)

A operação de junção interna é utilizada para realizar a combinação de linhas de uma tabela que se relacionam com linhas de outra tabela por meio da chave estrangeira.

Exemplo de Junção Interna

```
SELECT *

FROM Empregado INNER JOIN Departamento

ON Departamento_Codigo = Codigo;
```

Ao ser executada, a consulta apresentada gerará uma nova tabela composta por todos os atributos das duas tabelas com as linhas das duas tabelas que se relacionam por meio da coluna `Departamento_Codigo`, chave estrangeira da tabela `Empregado`, que referencia a coluna `Codigo` da tabela `Departamento`. A figura 5.13 apresenta a junção interna entre as tabelas `Empregado` e `Departamento`.

O caractere `*` na consulta representa todas as colunas das duas tabelas. É possível recuperar apenas um subconjunto das colunas: basta substituir o caractere `*` pelos nomes das colunas a serem recuperadas.

Matricula	Primeiro_Nome	Familia_Nome	Salario	Departamento_Codigo	Codigo	Nome	UF
123	Igor	Pereira	100.00	10	10	RH	RJ
159	Denise	Moreno	440.00	40	40	Financeiro	DF
369	Marcelo	Neiva	900.00	40	40	Financeiro	DF
456	Ana	Oliveira	200.00	20	20	TI	RJ
789	Clara	Silva	300.00	30	30	Logística	DF

Figura 5.13: Resultado do INNER JOIN entre as tabelas Empregado e Departamento.

Como pode ser observado na figura 5.13, os registros das tabelas `Empregado` e `Departamento` que compõem o resultado estão relacionados por meio dos campos `Departamento_Codigo` e `Codigo`. Os empregados que não estão associados a um departamento e os departamentos que não possuírem pelo menos um empregado associado não aparecem no resultado.

LEFT OUTER JOIN (Junção Externa à Esquerda)

A operação de junção externa à esquerda, assim como na junção interna, é utilizada para realizar a combinação de linhas de uma tabela que se relacionam com linhas de outra tabela por meio da chave estrangeira. Além disso, a junção externa à esquerda apresenta todos os registros da tabela cujo nome encontra-se à esquerda do comando de junção.

Exemplo de Junção Externa à Esquerda:

```
SELECT *

FROM Empregado  LEFT OUTER JOIN Departamento

ON Departamento_Codigo = Codigo;
```

Ao ser executada, a consulta apresentada gerará uma nova tabela composta por todos os atributos das duas tabelas com todas as linhas da tabela `Empregado`, por estar à esquerda do comando, combinadas às linhas da tabela `Departamento` que se relacionam por meio da coluna `Departamento_Codigo`, chave estrangeira da tabela `Empregado`, que referencia a coluna `Codigo` da tabela `Departamento`. A figura 5.14 apresenta a junção externa à esquerda entre as tabelas `Empregado` e `Departamento`.

O caractere `*` na consulta representa todas as colunas das duas tabelas, e é possível recuperar apenas um subconjunto das colunas: basta substituir o caractere `*` pelos nomes das colunas a serem recuperadas.

Matricula	Primeiro_Nome	Familia_Nome	Salario	Departamento_Codigo	Codigo	Nome	UF
123	Igor	Pereira	100	10	10	RH	RJ
159	Denise	Moreno	440	40	40	Financeiro	DF
369	Marcelo	Neiva	900	40	40	Financeiro	DF
456	Ana	Oliveira	200	20	20	TI	RJ
789	Clara	Silva	300	30	30	Logística	DF
963	Deyse	Silva	330	NULL	NULL	NULL	NULL

Figura 5.14: Resultado do LEFT OUTER JOIN entre as tabelas Empregado e Departamento.

Como pode ser observado na figura 5.14, os registros das tabelas `Empregado` e `Departamento` que compõem o resultado estão relacionados por meio dos campos `Departamento_Codigo` e `Codigo`. Os empregados que não estão associados a um departamento também aparecem no resultado, como é o caso do empregado de matrícula 963, porém neste caso os campos relativos à tabela `Departamento` aparecem com `null` (ausência de conteúdo).

RIGHT OUTER JOIN (Junção Externa à Direita)

A operação de junção externa à direita é idêntica à junção externa à esquerda. No entanto, essa operação apresenta todos os registros da tabela cujo nome encontra-se à direita do comando de junção, diferentemente da junção externa à esquerda que apresenta todos os registros da tabela cujo nome encontra-se à esquerda.

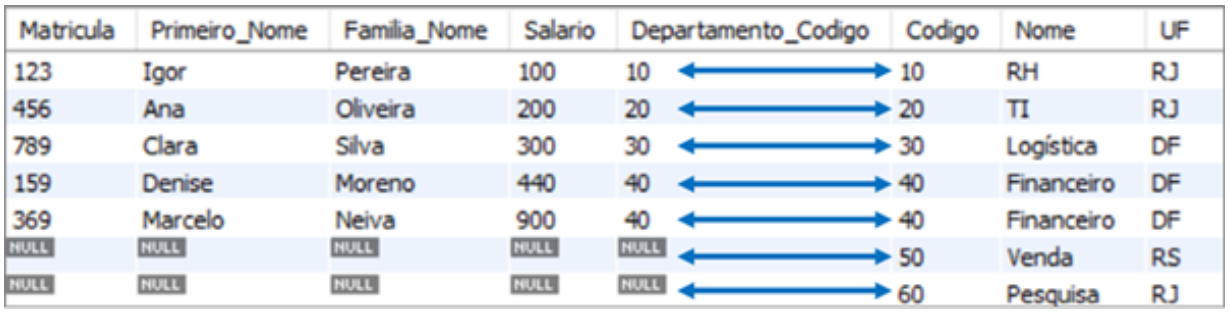
- **Exemplo de Junção Externa à Direita:**

```
SELECT *

FROM Empregado RIGHT OUTER JOIN Departamento

ON Departamento_Codigo = Codigo;
```

Ao ser executada, a consulta apresentada gerará uma nova tabela composta por todos os atributos das duas tabelas com todas as linhas da tabela `Departamento`, por estar à direita do comando, combinadas às linhas da tabela `Empregado` que se relacionam por meio da coluna `Departamento_Codigo`, chave estrangeira da tabela

`Empregado`, que referencia a coluna `Codigo` da tabela `Departamento`. A figura 5.15 apresenta a junção externa à direita entre as tabelas `Empregado` e `Departamento`.

O caractere `*` na consulta representa todas as colunas das duas tabelas, é possível recuperar apenas um subconjunto das colunas, basta substituir o caractere `*` pelos nomes das colunas a serem recuperadas.



Matricula	Primeiro_Nome	Familia_Nome	Salario	Departamento_Codigo	Codigo	Nome	UF
123	Igor	Pereira	100	10	10	RH	RJ
456	Ana	Oliveira	200	20	20	TI	RJ
789	Clara	Silva	300	30	30	Logística	DF
159	Denise	Moreno	440	40	40	Financeiro	DF
369	Marcelo	Neiva	900	40	40	Financeiro	DF
NULL	NULL	NULL	NULL	NULL	50	Venda	RS
NULL	NULL	NULL	NULL	NULL	60	Pesquisa	RJ

Figura 5.15: Resultado do RIGHT OUTER JOIN entre as tabelas Empregado e Departamento.

Como pode ser observado na figura 5.15, os registros das tabelas `Empregado` e `Departamento` que compõem o resultado estão relacionados por meio dos campos `Departamento_Codigo` e `Codigo`. Os departamentos que não estão associados a um empregado também aparecem no resultado como é o caso dos departamentos com os códigos 50 e 60, porém neste caso os campos relativos à tabela `Empregado` aparecem com `null` (ausência de conteúdo).

FULL OUTER JOIN (Junção Externa Completa)

A operação de junção externa completa é uma combinação da junção externa à esquerda com a junção externa à direita. Essa operação apresenta todos os registros das duas tabelas e combina as linhas das duas tabelas quando estão relacionadas.

- **Exemplo de Junção Externa Completa:**

```
SELECT *

FROM Empregado FULL OUTER JOIN Departamento

ON Departamento_Codigo = Codigo;
```

Ao ser executada, a consulta apresentada gerará uma nova tabela composta por todos os atributos das duas tabelas com todas as linhas das tabelas `Empregado` e `Departamento`. As linhas dessas duas tabelas são combinadas quando se relacionam por meio da coluna `Departamento_Codigo`, chave estrangeira da tabela `Empregado`, que referencia a coluna `Codigo` da tabela `Departamento`. A figura 5.16 apresenta a junção externa completa entre as tabelas `Empregado` e `Departamento`.

O caractere `*` na consulta representa todas as colunas das duas tabelas, é possível recuperar apenas um subconjunto das colunas, basta substituir o caractere `*` pelos nomes das colunas a serem recuperadas.

Matricula	Primeiro_Nome	Familia_Nome	Salario	Departamento_Codigo	Codigo	Nome	UF
123	Igor	Pereira	100	10	10	RH	RJ
159	Denise	Moreno	440	40	40	Financeiro	DF
369	Marcelo	Neiva	900	40	40	Financeiro	DF
456	Ana	Oliveira	200	20	20	TI	RJ
789	Clara	Silva	300	30	30	Logística	DF
963	Deyse	Silva	330	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	50	Venda	RS
NULL	NULL	NULL	NULL	NULL	60	Pesquisa	RJ

Figura 5.16: Resultado do RIGHT OUTER JOIN entre as tabelas Empregado e Departamento.

Como pode ser observado na figura 5.16, os registros das tabelas `Empregado` e `Departamento` que compõem o resultado estão relacionados por meio dos campos `Departamento_Codigo` e `Codigo`. Os empregados que não estão associados a um departamento também aparecem no resultado como é o caso do empregado de matrícula 963, porém neste caso os campos relativos à tabela `Departamento`

aparecem com `null` (ausência de conteúdo). Os departamentos que não estão associados a um empregado também aparecem no resultado como é o caso dos departamentos com os códigos 50 e 60, porém neste caso os campos relativos à tabela `Empregado` aparecem com `null` (ausência de conteúdo).

Após a explicação dos vários tipos de junção, finalizamos o aprendizado sobre modelagem de dados e linguagem SQL. No entanto, cabe ressaltar que a linguagem SQL possui vários recursos além dos apresentados neste livro, porém optamos por apresentar apenas um resumo da linguagem SQL para que o leitor possa realizar as operações básicas com dados armazenados em tabelas relacionais.

Apesar do modelo relacional ainda ser muito utilizado, com o advento do Big Data, houve a necessidade da implementação de novos modelos de banco de dados capazes de lidar com dados não estruturados e altamente distribuídos, pois os bancos de dados relacionais apresentavam problemas de performance nesse ambiente. Nesse contexto, surgiram os bancos de dados NoSQL.

No próximo capítulo apresentaremos brevemente as características do Big Data e os bancos de dados NoSQL para que você possa conhecer as demandas desse ambiente e outros paradigmas de banco de dados, além do relacional.

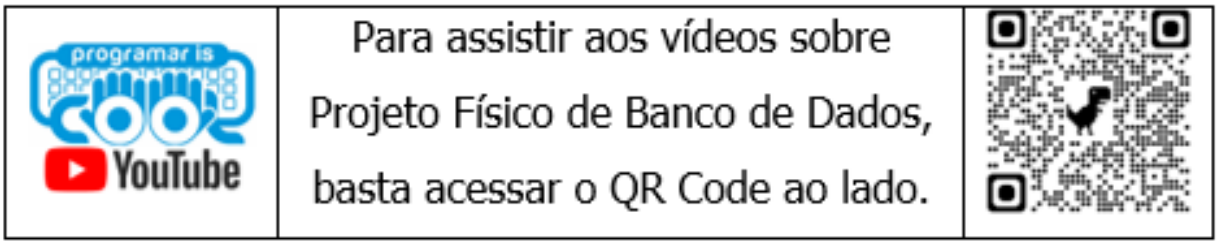


Figura 5.17: Vídeos sobre Projeto Físico de Banco de Dados. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIDNlcKPp_EcQIT5E_oozy_M

Referências

SILBERSCHATZ, Abraham, KORTH, Henry F. e SUDARSHAN, S. *Sistema de Banco de Dados*. Editora Campus. 5a Edição, 2006.

Capítulo 6
Big Data e banco de dados NoSQL

Estima-se que, do início da civilização até 2003, a humanidade tenha criado cinco exabytes (10 bytes elevados à 18ª potência) de dados. No início da década de 2010, criava-se esse mesmo volume de dados a cada dois dias (SCHMIDT, 2010). De acordo com a ONU (https://news.un.org/pt/story/2022/09/1801381), 66% da população mundial tinha acesso à internet no ano de 2022, ou seja, aproximadamente 5,3 bilhões de pessoas. O órgão ainda ressalta que medidas devem ser tomadas para que 2,7 bilhões de pessoas que nunca tiveram acesso a rede possam se conectar, isto é, para que 8 bilhões de pessoas possam acessar a internet.

Vailshery (2024), em seu artigo publicado no Statista (www.statista.com), um portal online de estatística que disponibiliza dados recolhidos por institutos de pesquisa de mercado, afirma que a expectativa era a criação global de 97 zettabytes (10 bytes elevados a 21ª potência) em 2022. Além disso, a projeção é de que, até 2025, esse número alcance a marca de 181 zettabytes.

Algumas explicações para tal fenômeno são a drástica redução de preços para o armazenamento das informações; a explosão de aplicações disponíveis na internet (*e-commerce*), a popularização de sensores conectados (Internet das Coisas), pesquisas científicas (Projeto Genoma) e as redes sociais.

Tal cenário demanda soluções efetivas em termos de custos e formas inovadoras de tratamento da informação para uma melhor percepção e tomada de decisão. Uma das propostas emergentes para lidar com esse ambiente complexo é o ***Big Data***.

6.1 Big Data

Há várias definições, entendimentos e discussões para o termo "*Big Data*", uma das mais aceitas é a definição de **3Vs**, apresentada por Laney (2001). O *Big Data* é caracterizado por um grande **volume** de dados, adquiridos e processados em uma **velocidade** apropriada para não se tornarem obsoletos e disponibilizados em uma grande gama de **variedade** de formatos.

Com o passar do tempo, no entanto, novos trabalhos sobre *Big Data* foram surgindo, e, consequentemente, novas proposições foram sendo firmadas a partir dos 3 Vs iniciais de Laney. Gandomi e Haider (2015), por exemplo, acrescentam mais três dimensões: **veracidade**, **variabilidade** e **valor**. Segundo os autores, a veracidade está relacionada com a insegurança inerente que algumas fontes de dados possuem — ou seja, lidar com dados imprecisos e incertos pode ser outra faceta do *Big Data*; por sua vez, a variabilidade está relacionada com a variação nas taxas de fluxo dos dados — esta dimensão relaciona-se com a velocidade de produção de dados, podendo haver alta e baixa velocidade (variabilidade); por último, a característica valor é considerada um atributo definidor do *Big Data* — os dados recebidos em sua forma original geralmente possuem baixo valor em relação ao seu volume, podendo ser gerado alto valor a partir da análise de grandes volumes desses dados.

Davenport (2014) afirma que o *Big Data* é inegavelmente grande. Por outro lado, possui uma designação um tanto quanto inapropriada, pois se trata de um termo genérico para dados que não cabem em repositórios habituais. Segundo o autor, o *Big Data* se refere a dados massivamente volumosos para caberem em simples servidores, extremamente desestruturados para se ajustarem a bancos de dados com base em linhas e colunas de tabelas relacionais, e continuamente fluídos para caberem em estruturas estáticas de armazenagem.

A partir das definições apresentadas, pode-se inferir que o principal objetivo do *Big Data* é proporcionar um ambiente apropriado à análise de grandes volumes de dados distribuídos para a geração de informações valiosas e, consequentemente, novas ideias (*insights*) que vão auxiliar as organizações no momento da tomada de decisão. Essa abordagem gera desafios oriundos dessa proliferação de dados, refletindo, assim, na superação da capacidade de processamento das tecnologias tradicionais.

Para lidar com esses desafios, uma nova geração de arquiteturas foi proposta, entre elas, o conceito de *MapReduce* apresentado pelos pesquisadores da empresa Google, Jeffery Dean e Sanjay Ghemawat (2008), implementado pelo Hadoop. O projeto Apache Hadoop nasceu da necessidade de processar de forma mais rápida e confiável os dados do *Big Data*. Trata-se de um ecossistema de software de código aberto utilizado para armazenar e analisar os dados do *Big Data*. Em vez de depender de hardware para fornecer alta disponibilidade crítica, a natureza distribuída do Hadoop é projetada para detectar e lidar com falhas na camada de aplicativo, oferecendo um serviço altamente disponível em cima de um cluster (conjunto) de computadores para reduzir os riscos de falhas de máquinas independentes (GoogleCloud, 2024).

Em substituição a usar um grande computador (escalabilidade vertical) para armazenar e processar dados, o Hadoop usa clusters de vários computadores (escalabilidade horizontal) para analisar conjuntos de dados massivos em paralelo. O Hadoop pode lidar com várias formas de dados estruturados e não estruturados, o que proporciona maior velocidade e flexibilidade para coletar, processar e analisar dados em relação aos bancos de dados relacionais (GoogleCloud, 2024).

O projeto Apache Hadoop foi desenvolvido pela organização sem fins lucrativos *Apache Software Foundation* (ASF). No entanto, como o programa foi desenvolvido em código aberto, surgiram várias outras versões: Cloudera Hadoop, Hortonworks Hadoop, EMC Hadoop, Intel Hadoop, Microsoft Hadoop etc.

O MapReduce, implementado pelo Hadoop, pode ser dividido em dois estágios principais:

- **Mapeamento:** o nó mestre divide os dados em vários subconjuntos menores, um nó trabalhador processa um subconjunto de dados menor sob o controle de um rastreador de trabalho e armazena o resultado no sistema de arquivos local, onde um redutor será capaz de acessá-lo.
- **Redução:** esta etapa analisa e reúne os dados de entrada a partir das etapas de mapeamento. Pode haver múltiplas tarefas de redução para paralelizar a agregação, e essas tarefas são executadas nos nós trabalhadores sob o controle do rastreador de trabalho.

A figura 6.1 apresenta uma visão macro dos principais estágios do MapReduce.

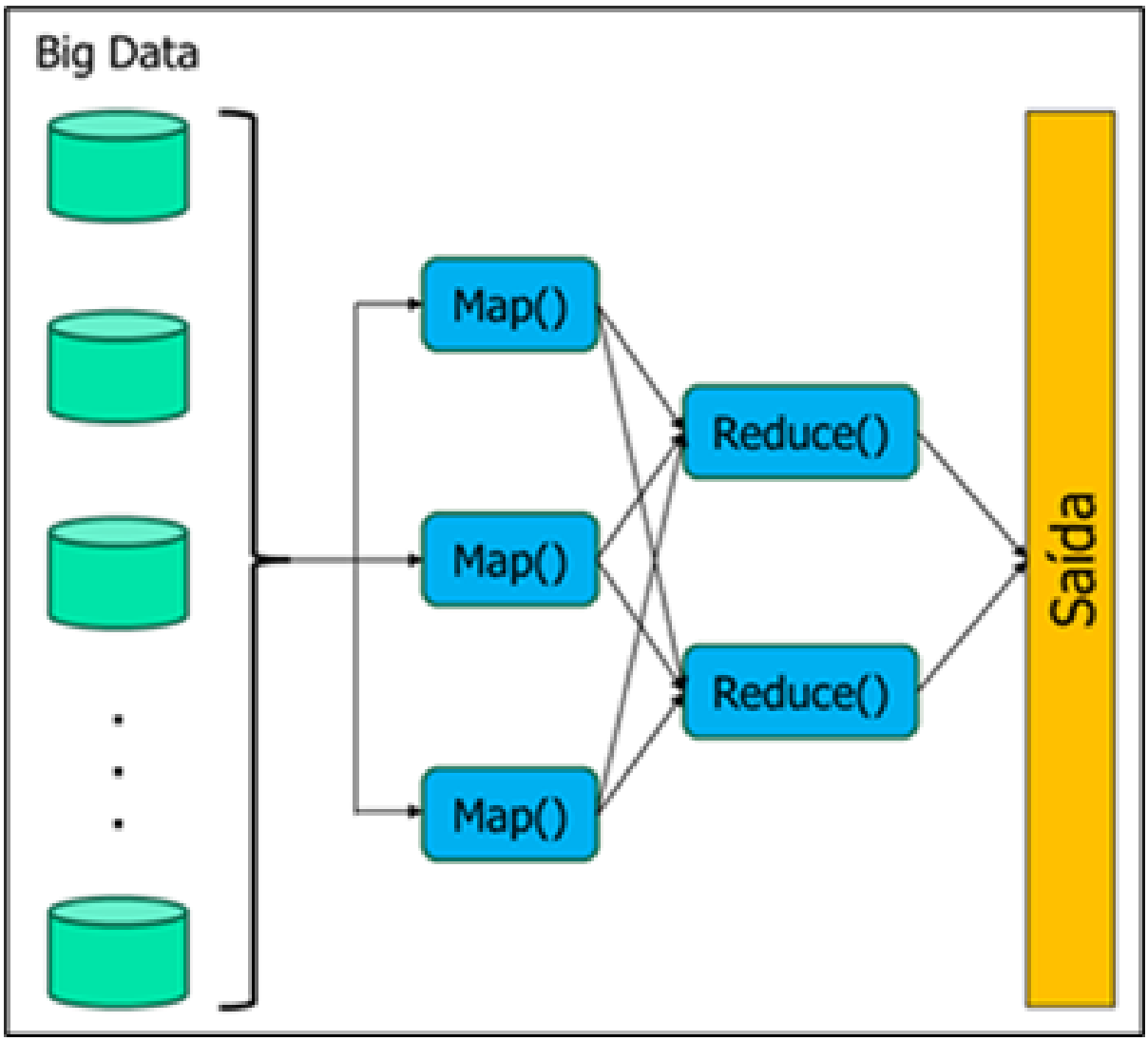


Figura 6.1: Principais estágios do MapReduce.

À esquerda da figura 6.1, podem ser observadas as fontes de dados distribuídas e heterogêneas que compões o *Big Data*. Percorrendo essa figura da esquerda para a direita, pode-se observar a camada de mapeamento, representada pelos processos `Map()`, e a camada de redução, composta pelos processos `Reduce()`. Na verdade, o mapeamento consiste em encontrar e preparar os dados para a redução. A redução consiste em aplicar as regras de negócio, como sumarizações, aos dados.

Como exemplo de utilização do MapReduce, poderíamos supor o seu uso pelo site da Amazon para contabilizar o total de vendas diárias

de eletrônicos. A Amazon possui milhares de servidores e repositórios de dados distribuídos com o registro de suas vendas. Então, o primeiro passo para totalizar as vendas diárias de eletrônicos seria localizar as notas fiscais com itens eletrônicos, obter esses itens com os respectivos valores e organizar esses dados em um local acessível, essa é a tarefa do mapeamento do MapReduce. Em seguida, a redução consistiria em acessar esse local onde se encontram os itens eletrônicos com os respectivos valores e calcular o somatório de vendas diárias.

A fim de tornar mais claro o processamento MapReduce, a figura 6.2 apresenta o seu uso para contar o número de ocorrências de palavras exclusivas em um arquivo texto.

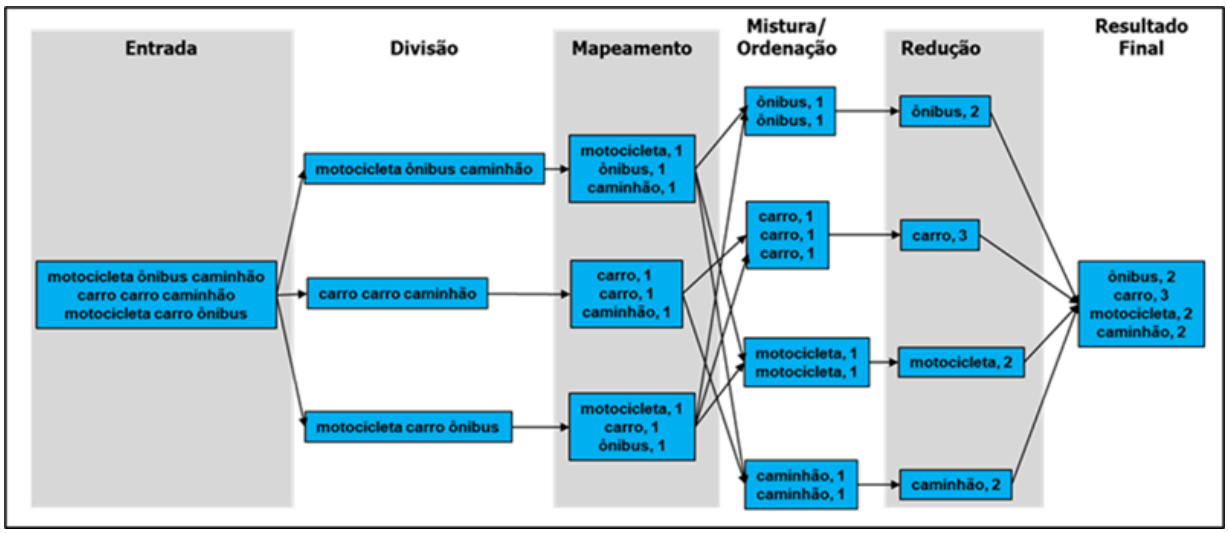


Figura 6.2: O uso do MapReduce para contar o número de ocorrências de palavras exclusivas em um arquivo texto.

À esquerda da figura 6.2, pode ser observado um arquivo texto com várias palavras. Percorrendo essa figura da esquerda para a direita, pode-se observar os passos da execução do MapReduce para contar o número de ocorrências de cada palavra contida nesse arquivo.

Inicialmente, o conjunto de palavras contido no arquivo é dividido em partes menores para maior eficiência. Nesse caso, a entrada é dividida em 3 partes que são chamados de divisões de entrada na

coluna divisão. Como existem três divisões de entrada, três mapeadores diferentes serão usados na coluna mapeamento, um para cada divisão.

Cada mapeador analisa uma linha, obtém cada palavra e define uma estrutura chave-valor da seguinte forma: `<palavra, 1>`. Neste exemplo, a saída do mapeador para a linha `<motocicleta ônibus caminhão>`, será `<motocicleta, 1> <ônibus, 1> <caminhão, 1>`. Assim, todos os mapeadores terão uma chave como palavra e o valor 1.

Na próxima coluna, a mistura e ordenação, a saída do par chave-valor do mapeador com a mesma chave será consolidada. A chave produzida pelos mapeadores será comparada e classificada.

A seguir, a coluna redução receberá a lista chave-valor organizada como entrada, contará apenas o número ocorrências do valor 1 na lista e definirá o valor de contagem como saída. A atividade de redução nesse caso é muito simples, consiste apenas na contagem das ocorrências de cada palavra, mas poderia ser uma atividade muito mais complexa, como ocorre normalmente em casos reais.

No final, todos os pares chave-valor de saída da fase redutor serão consolidados em um arquivo como saída final. Essa é a estratégia de processamento do Hadoop.

Cabe ressaltar que novos frameworks foram propostos para substituírem ou serem usados em conjunto com o Hadoop, porém o aprendizado da arquitetura MapReduce é de vital importância para entendimento do processamento dos dados em um ambiente de *Big Data*.

O *Big Data*, como já abordado anteriormente, possui como uma de suas características a variedade de formatos de seus dados. Isso significa que é composto tanto por dados estruturados — aqueles que podem ser armazenados em bancos de dados convencionais (linhas e colunas) —, quanto por dados não estruturados — dados que não preenchem os requisitos para armazenagem em bancos de dados relacionais. Focando nessa característica, surge uma nova

geração de bancos de dados, projetados para serem distribuídos em vários nós, sem esquemas rígidos, baseados em modelos não-relacionais, chamados bancos de dados **NoSQL**.

6.2 Banco de dados NoSQL

Os maiores desafios dos bancos de dados NoSQL é armazenar, recuperar e processar, de forma eficaz, a grande quantidade de dados estruturados e não estruturados disponíveis no ambiente de *Big Data*. Esse ambiente gerou novas necessidades, tais como:

- Processar o volume crescente de dados;
- Esquemas flexíveis;
- Maior distribuição do processamento;
- Flexibilização das estruturas de armazenamento desses dados;
- Escalabilidade horizontal; e
- Lidar com vários formatos de documentos.

O termo "NoSQL" foi utilizado pela primeira vez em 1990 por Carlo Strozzi para designar um BD relacional de código aberto que armazenava suas tabelas sob a forma de arquivos ASCII, e cada tupla era representada por uma linha com os campos separados por tabulação. O nome vem do fato de que o BD não utilizava SQL como linguagem de consulta. Em vez disso, ele era manipulado por *shell scripts*.

O uso do termo "NoSQL" como é utilizado hoje é resultado de uma reunião realizada no dia 11 de julho de 2009, em São Francisco, organizada por Johan Oskarsson. "NoSQL" surgiu como um *hashtag* para o Twitter por ser curto, fácil de lembrar e por não existirem muitos semelhantes. Oskarsson não esperava que se tornasse o nome da tendência tecnológica.

Apenas em 2011 esse termo passou a representar uma família de bancos de dados que podem ser categorizados nos seguintes

modelos: par chave/valor, documentos, família de colunas e grafos (FOWLER; SADALAGE, 2019).

As principais características dos SGBDs NoSQL são:

- Não utilizam o SQL (possuem linguagens semelhantes);
- Geralmente são projetos de código aberto;
- A maioria dos BDs é orientada pela necessidade de execução distribuída;
- Oferecem uma gama de opções para consistência e distribuição;
- Não possuem esquemas rígidos.

Banco de Dados NoSQL Modelo Chave-Valor

Um banco de dados modelo chave-valor utiliza um método de chave-valor simples para armazenar dados. Armazena dados como um conjunto de pares de chave-valor em que uma chave funciona como um identificador exclusivo que identifica o valor que pode ser qualquer conteúdo.

Esse tipo de banco de dados é utilizado quando todo o acesso ao banco de dados é feito por meio de um identificador único. As operações de inserção e atualização de dados são feitas da seguinte forma: dado um par chave-valor, se a chave já existir no banco de dados, o valor atual é sobrescrito; caso contrário, cria-se um registro. A figura 6.3 apresenta a estrutura de um banco de dados chave-valor.

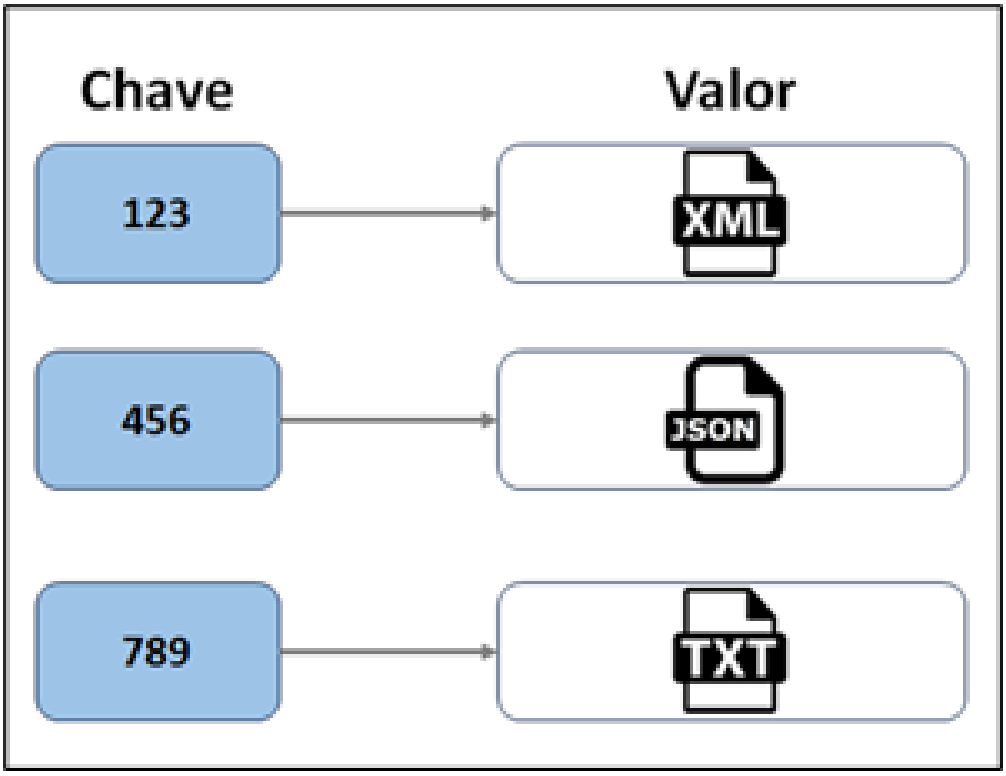


Figura 6.3: Banco de dados modelo chave-valor.

Como pode ser observado na figura 6.3, os bancos de dados chave-valor são os depósitos de dados NoSQL mais simples de utilizar. O valor é um dado que o depósito apenas armazena, sem se preocupar ou saber seu conteúdo. É responsabilidade do aplicativo entender o que foi armazenado. O valor pode ser um conteúdo em XML, JSON, texto etc. O acesso sempre é feito pela chave e, por isso, esses bancos de dados geralmente apresentam um ótimo desempenho e escalabilidade.

Banco de Dados NoSQL Modelo Documentos

Um banco de dados modelo documentos armazena documentos apontados por identificadores, ou seja, esses bancos de dados são estruturados para armazenarem várias ocorrências de pares formados por um identificador e um conteúdo, por isso são semelhantes aos bancos de dados chave-valor. No entanto, diferentemente do banco de dados modelo chave-valor, o banco de

dados modelo documentos armazena documentos que podem ser consultados pelos seus conteúdos. Esses documentos armazenados podem possuir estruturas diferentes.

Esse tipo de banco de dados foi projetado para armazenar e consultar dados como documentos do tipo JSON. A natureza flexível, semiestruturada e hierárquica dos documentos e dos bancos de dados de documentos permite que eles evoluam conforme as necessidades dos aplicativos. A figura 6.4 apresenta a estrutura de um banco de dados de documentos.

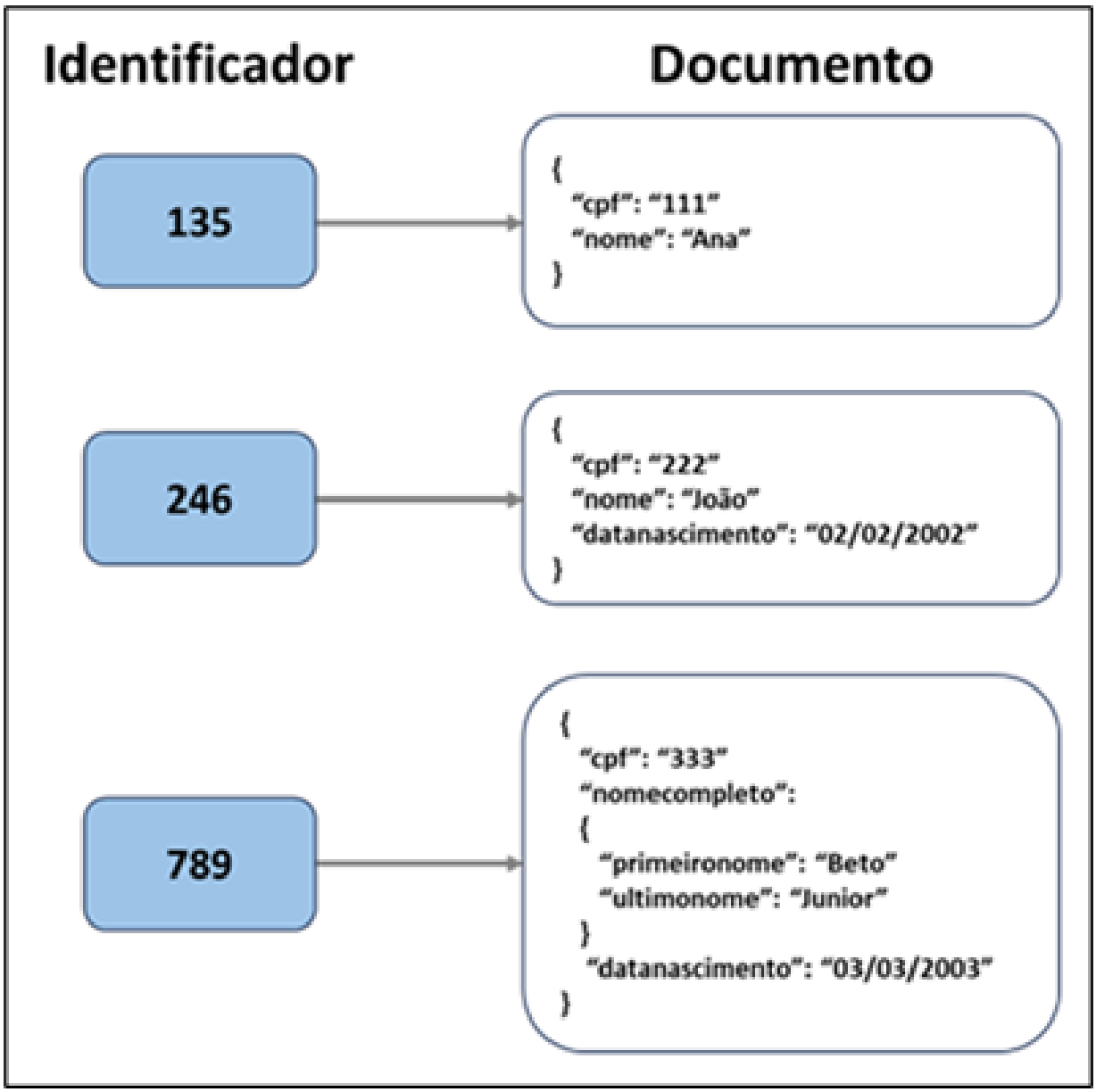


Figura 6.4: Banco de dados modelo documentos.

A figura 6.4 apresenta três documentos JSON com os respectivos identificadores armazenados em um banco de dados de documentos. É possível observar que, embora todos os documentos sejam no formato JSON, suas estruturas são diferentes. O primeiro documento possui os atributos `cpf` e `nome` ; o segundo apresenta os atributos `cpf` , `nome` e `datanascimento` ; finalmente, o terceiro documento apresenta os atributos `cpf` , `nomecompleto` e

`datanascimento`, sendo que o atributo `nomecompleto` é composto pelos atributos `primeironome` e `ultimonome`.

O banco de dados de documentos funciona bem com casos em que cada documento é único e evolui com o passar do tempo. Esses documentos armazenados, considerando o mesmo tipo de arquivo, são semelhantes entre si, mas não precisam possuir exatamente a mesma estrutura.

Banco de Dados NoSQL Modelo Família de Colunas

Nos bancos de dados modelo família de colunas, cada família de colunas pode ser comparada a um conjunto de linhas em uma tabela de um banco de dados relacional, onde a chave primária identifica a linha, que é constituída de múltiplas colunas. A figura 6.5 apresenta a estrutura de um banco de dados família de colunas.

CPF	Nome	Sobrenome	Nascimento	Pais
111	Ana	Silva	01/01/2001	
222	Beto	Lima		
333	Caio	Abreu	03/02/2003	Brasil
444	Lia	Correia		

Figura 6.5: Banco de dados modelo família de colunas.

Como pode ser observado na figura 6.5, diferentemente do banco de dado relacional, no qual todas as linhas de uma mesma tabela possuem as mesmas colunas, no banco de dados família de colunas, linhas diferentes não precisam possuir as mesmas colunas. Nessa figura, a linha cujo CPF é 111 possui as colunas `CPF`, `Nome`, `Sobrenome` e `Nascimento`. Já as linhas com os CPFs 222 e 444 possuem apenas as colunas `CPF`, `Nome` e `Sobrenome`. Finalmente, a linha cujo CPF é 333 possui as colunas `CPF`, `Nome`, `Sobrenome`, `Nascimento` e `Pais`.

Cabe ressaltar que, nesses bancos de dados, novas colunas podem ser adicionadas a qualquer linha, a qualquer momento, sem a obrigatoriedade de ter que adicioná-las às outras linhas também.

Banco de Dados NoSQL Modelo Grafos

Os bancos de dados modelo grafos permitem a criação de entidades com foco em seus relacionamentos. São compostos por nós e arestas: usam nós para armazenar entidades de dados e arestas para armazenar os relacionamentos entre as entidades. Uma aresta tem sempre um nó inicial, um nó final, um tipo e um direcionamento, o que possibilita a descrição dos relacionamentos entre os nós.

Com estes bancos é possível construir redes complexas de relacionamentos e revelar novos dados através da inferência. A figura 6.6 apresenta a estrutura de um banco de dados de grafos.

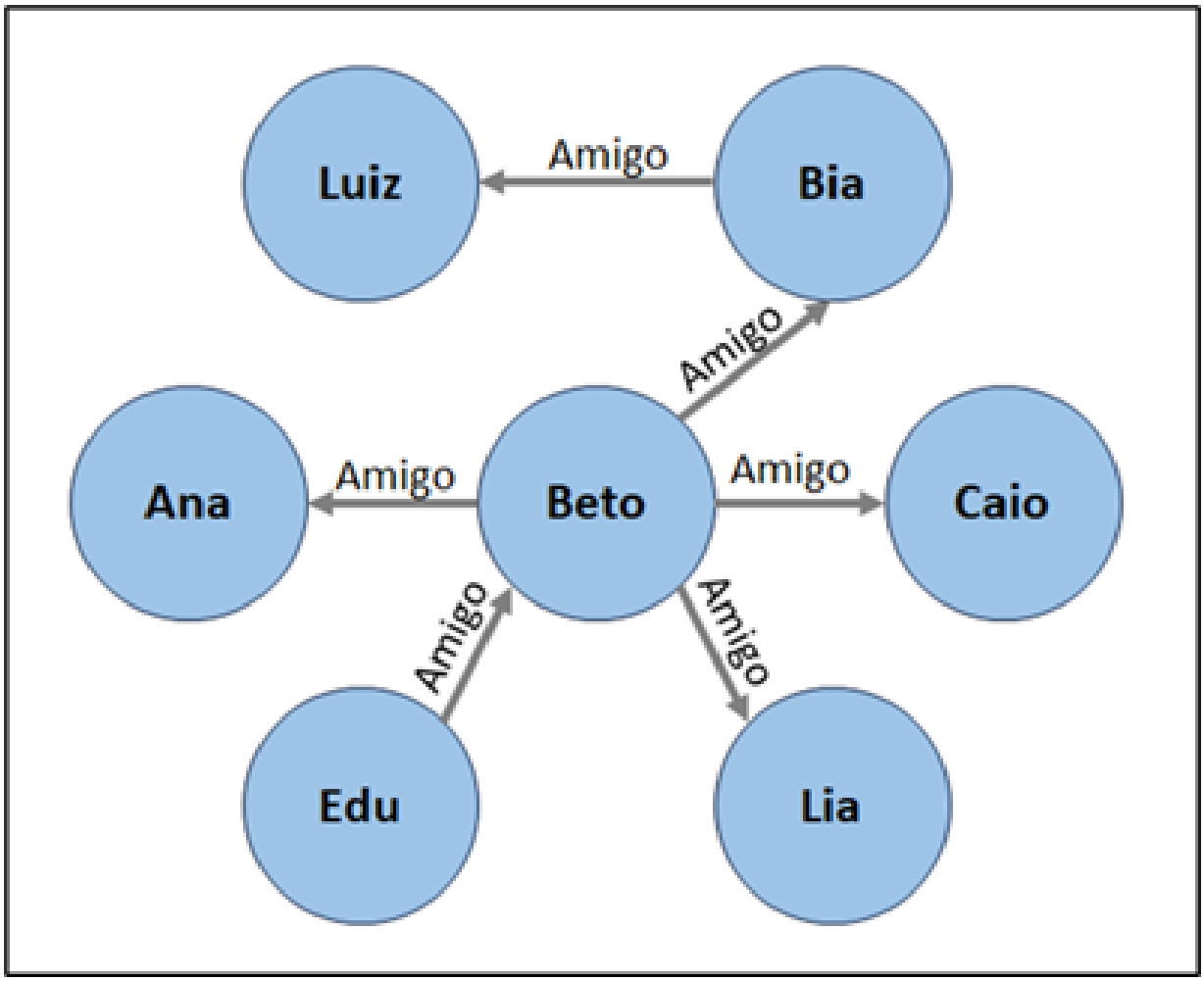


Figura 6.6: Banco de dados modelo grafos.

Como pode ser observado na figura 6.6, em um banco de dados de grafos, uma aresta tem sempre um nó inicial, um nó final, um tipo e um direcionamento, o que possibilita a descrição dos relacionamentos entre dois nós. A quantidade e os tipos de relacionamentos que um nó pode ter são ilimitados.

Esses bancos de dados são vantajosos em casos de uso como redes sociais; porém, por operarem em nós conectados, a maioria desses bancos de dados não suporta a distribuição dos nós em servidores diferentes e, por serem orientados a relacionamentos, sua fragmentação é difícil.

Após a apresentação sucinta dos tipos de bancos de dados NoSQL, conclui-se que as principais vantagens desses bancos são: a flexibilidade de suas estruturas, a alta escalabilidade horizontal, o suporte a dados não estruturados e o processamento distribuído. Tais características tornam os bancos de dados NoSQL excelentes dispositivos de persistência para o ambiente de *Big Data*.

Para finalizar este capítulo, a figura 6.7 apresenta uma arquitetura composta pelo Hadoop e bancos de dados NoSQL em um ambiente de *Big Data*.

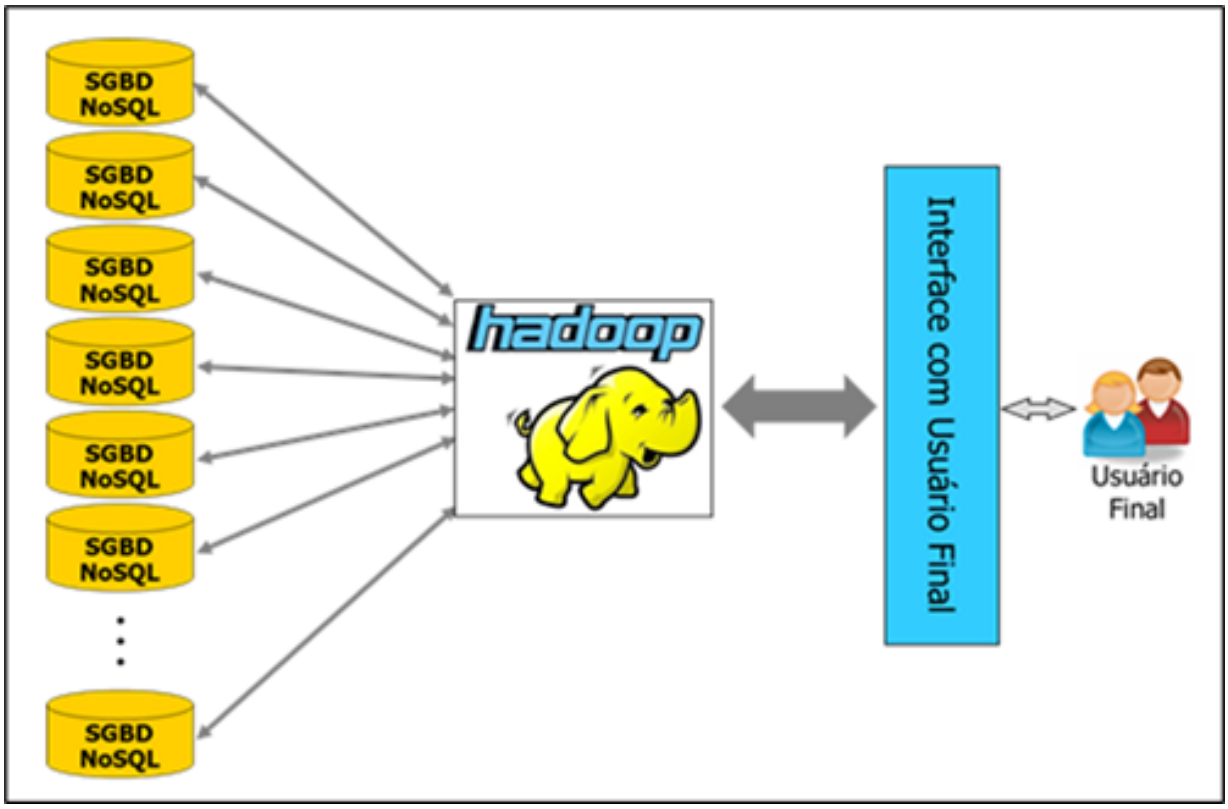


Figura 6.7: Arquitetura composta pelo Hadoop e bancos de dados NoSQL.

Na arquitetura apresentada na Figura 6.7, os dados são persistidos em centenas ou milhares de bancos de dados NoSQL. O Hadoop age como um middleware, ou seja, um software intermediário que, após mapear os dados distribuídos e aplicar as transformações ou regras de negócio (redução), consolida o resultado e apresenta a resposta aos usuários finais. Com essa arquitetura, torna-se possível o

processamento, a uma velocidade apropriada, de um enorme volume de dados, distribuídos e dispostos nos mais variados formatos, atendendo, assim, às demandas do *Big Data*.

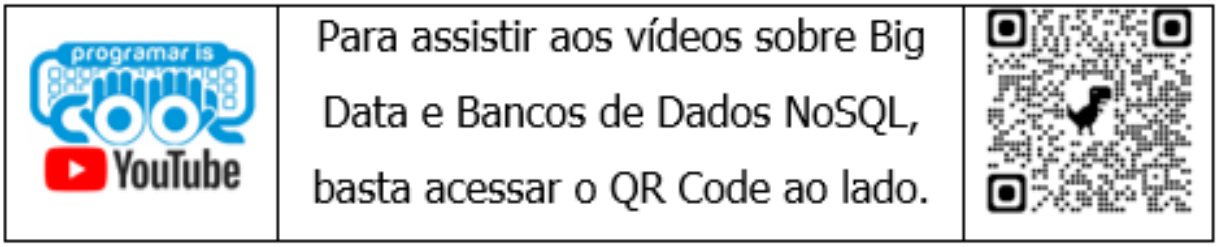


Figura 6.8: Vídeos sobre Big Data e Banco de Dados NoSQL. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIBp_y7se8FxGvJ4TVnzH6TM

Referências

DAVENPORT, T. H. *Big Data at work: dispelling the myths, uncovering the opportunities*. Harvard Business Review Press, 2014.

DEAN, J.; GHEMAWAT, S. *MapReduce: simplified data processing on large clusters*. Communications of the ACM, v. 51, n. 1, p. 107-113, 2008.

GANDOMI, A.; HAIDER, M. Beyond the hype: big data concepts, methods, and analytics. *International Journal of Information Management*, [S.l.], v. 35, n. 2, p. 137-144, Apr. 2015.

GoogleCloud. *O que é o Apache Hadoop?* 2024. Disponível em: https://cloud.google.com/learn/what-is-hadoop?hl=pt-

br#:~:text=O%20Apache%20Hadoop%20nasceu%20da,armazenar%20e%20analisar%20Big%20Data. Acesso em: 4 mai. 2024.

LANEY, D. *Application delivery strategies*. Meta Group.,2001.

FOWLER, Martin; SADALAGE, Pramod J. *NoSQL essencial: um guia conciso para o mundo emergente da persistência poliglota*. Novatec Editora, 2019.

SCHMIDT, E. Every 2 days we create as much information as we did up to 2003. *TechCrunch*, 2010.

VAILSHERY, L. S. *Edge computing market revenue worldwide from 2019 to 2026*. Statista, 2024. Disponível em: https://www.statista.com/statistics/1175706/worldwide-edge-computing-market-revenue/. Acesso em: 02 mai. 2024.

CAPÍTULO 7
Data Warehouse

De acordo com Edwards Deming, considerado o pai da melhoria contínua da qualidade, "*sem os dados, você é apenas mais uma pessoa com opinião*". No ambiente corporativo, principalmente nas empresas orientadas a dados, a pessoa com poder de decisão deve tomar decisões com base em dados internos e externos à organização, não basta apenas ter uma opinião sobre o assunto. Atualmente, dados são gerados e disponibilizados a todo momento pelos mais diversos tipos de organizações públicas ou privadas; entretanto, na maioria dos casos, os dados de interesse da organização não estão integrados e organizados no formato apropriado para o apoio à decisão. Para se tornarem relevantes, eles precisam ser tratados.

Uma estratégia muito utilizada para coletar, transformar, armazenar e apresentar os dados para o apoio à decisão é a criação de um ambiente de **Data Warehouse** (DW) organizado em uma arquitetura OLAP, cuja estrutura já foi apresentada no capítulo 2, no tópico *Sistemas de apoio à decisão*.

Neste capítulo, abordaremos a técnica de *modelagem dimensional* para você entender como os dados são modelados para serem armazenados no DW.

7.1 Modelagem dimensional

A **modelagem dimensional** é uma técnica de projeto lógico para estruturar os dados de maneira que sejam intuitivos para os usuários do negócio e para que forneçam um alto desempenho de acesso aos dados (KIMBALL, 2008). A fim de fornecer subsídios aos relatórios

de negócio e às necessidades analíticas, a modelagem dimensional requer uma abordagem diferente daquelas usadas no projeto de bancos de dados transacionais. O modelo dimensional é composto de dois tipos de tabelas: *fato* e *dimensão*.

A tabela **fato** é a tabela central do modelo dimensional. Ela armazena medições numéricas do negócio, na qual cada uma das medições é obtida pela interseção das dimensões. Ela possui chaves de várias dimensões e deve conter o *grão* (nível de detalhamento) mais detalhado, capturado a partir do processo de negócio. Em consultas à tabela *fato* são usados centenas, milhares ou até milhões de registros para a construção da resposta.

Por outro lado, a tabela **dimensão** é uma entidade independente em um modelo dimensional que serve como mecanismo para aplicar o recurso de separação e combinação das métricas da tabela *fato* em um modelo dimensional. Normalmente possui uma chave primária artificial simples (*surrogate key*); é pequena (se comparada à tabela *fato*) e armazena dados descritivos do negócio. A chave primária simples da tabela *dimensão* corresponde exatamente à chave estrangeira da tabela *fato*, o que permite a ligação entre as tabelas.

Os modelos dimensionais armazenados em bancos de dados relacionais são chamados de esquemas *estrela* (*star*) ou *flocos de neve* (*snowflake*); nesse caso, a arquitetura OLAP é denominada ROLAP (*Relational Online Analytical Processing*). Por outro lado, modelos dimensionais armazenados em bancos de dados multidimensionais são chamados de *cubos*; nesse caso, a arquitetura OLAP é denominada MOLAP (*Multidimensional Online Analytical Processing*). Quando um modelo multidimensional possui mais de três dimensões, é denominado *hipercubo*. Existe também a arquitetura HOLAP (*Hybrid Online Analytical Processing*), cujo repositório de dados possui características relacionais e multidimensional simultaneamente.

A figura 7.1 apresenta um modelo estrela com uma tabela fato relacionada a quatro tabelas do tipo dimensão:

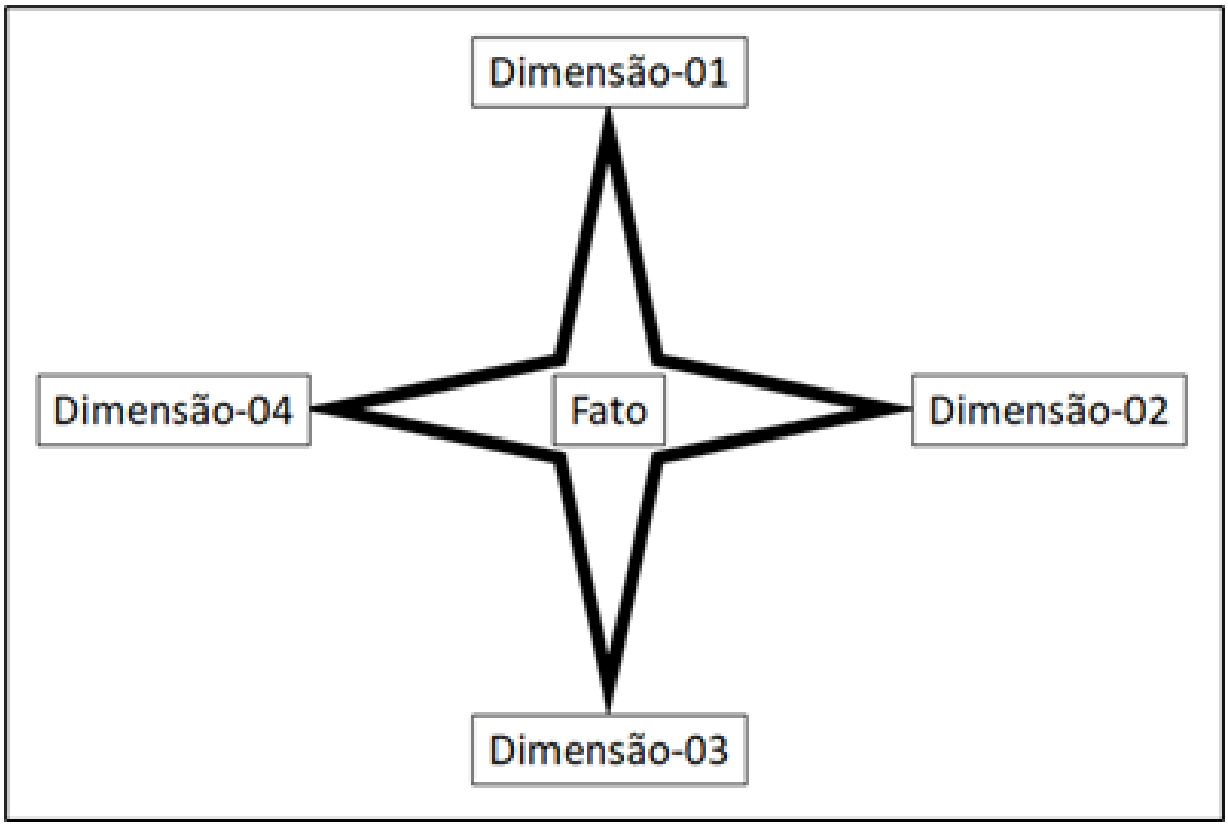


Figura 7.1: Modelo estrela.

O exemplo de modelo estrela apresentado na figura 7.1 possui apenas quatro dimensões, porém, em projetos reais, esses modelos costumam possuir dezenas de dimensões. As dimensões descrevem as características do negócio, enquanto o fato quantifica a combinação dessas características.

Esse modelo estrela, no nível conceitual, pode ser representado por meio do modelo entidade-relacionamento, que vimos no capítulo 3. A figura 7.2 apresenta a representação do modelo estrela da figura 7.1 utilizando a simbologia do diagrama entidade-relacionamento da engenharia da informação:

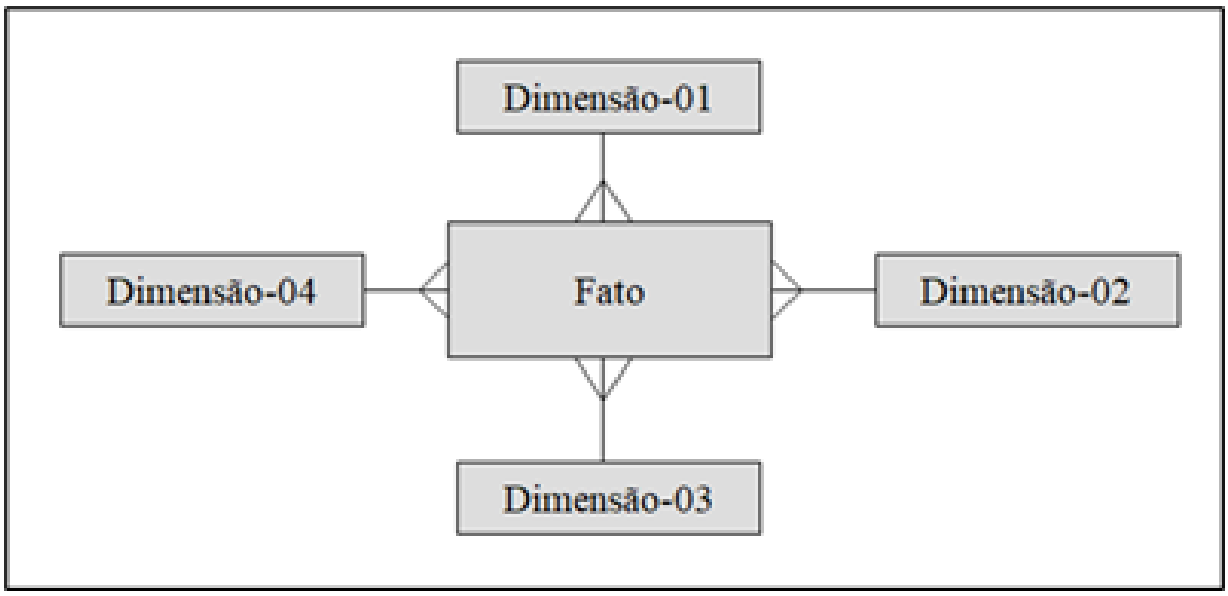


Figura 7.2: Representação do modelo estrela por meio do modelo entidade-relacionamento.

O modelo entidade-relacionamento representado na figura 7.2 é composto por cinco entidades: a entidade central representa o fato e as periféricas são as dimensões. É possível observar que existe um relacionamento identificador de cardinalidade `1:n` entre cada dimensão e a tabela fato.

Assim, quando essas entidades forem mapeadas para tabelas, a tabela *fato* será composta por, no mínimo, quatro colunas, e cada uma delas será uma chave estrangeira que referenciará a chave primária de cada dimensão. O conjunto dessas quatro chaves estrangeiras comporá a chave primária da tabela *fato*. Por isso, a tabela *fato* é considerada uma variável dependente, pois depende das dimensões para existir, e as dimensões são consideradas variáveis independentes.

A figura 7.3 apresenta o modelo lógico relacional resultante do mapeamento do modelo conceitual apresentado na figura 7.2:

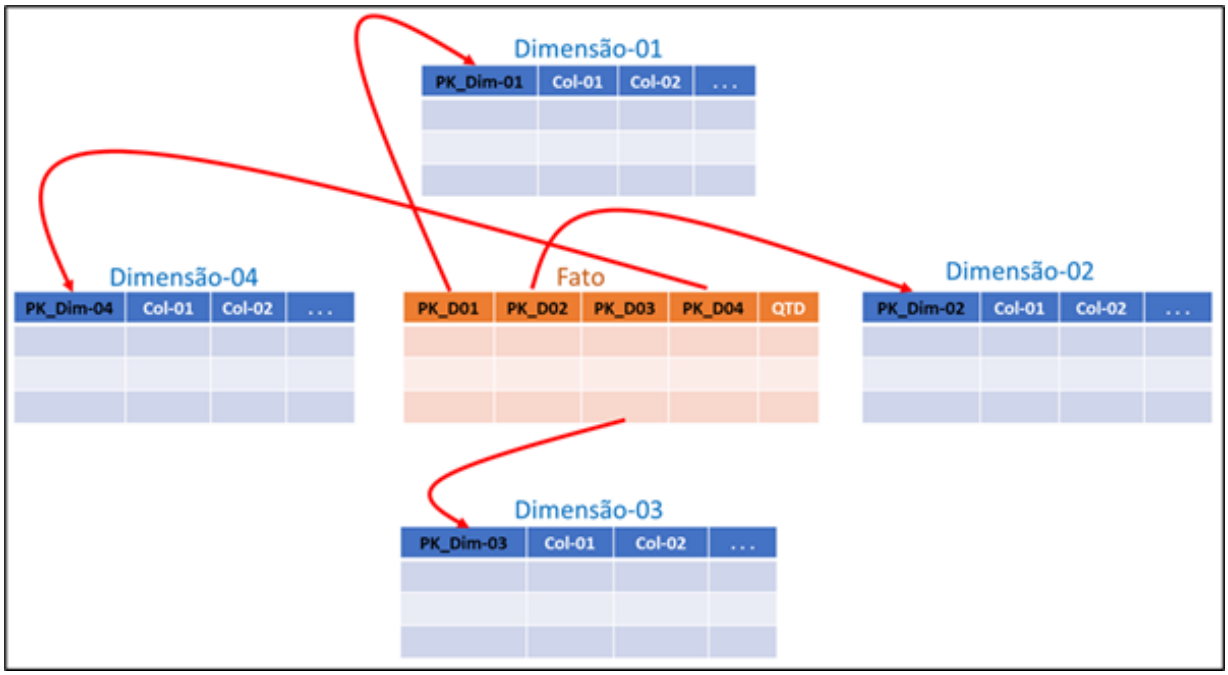


Figura 7.3: Representação do modelo estrela no nível lógico relacional.

Na figura 7.3, é possível observar uma tabela *fato*, denominada `Fato`, e quatro tabelas do tipo dimensão. As dimensões são a `Dimensão-01`, a `Dimensão-02`, a `Dimensão-03` e a `Dimensão-04`, e suas respectivas colunas de chaves primárias são `PK_Dim-01`, `PK_Dim-02`, `PK_Dim-03` e `PK_Dim-04`. Essas dimensões possuem outras colunas além da chave primária, que são denominadas `Col-01`, `Col-02` e demais colunas representadas pelas reticências.

Nessa figura, a tabela *fato* é formada pelas colunas `PK_D01`, `PK_D02`, `PK_D03`, `PK_D04` e `QTD`. As quatro primeiras colunas juntas compõem a chave primária da tabela *fato*, sendo que cada uma delas separadamente é uma chave estrangeira que referencia uma dimensão específica. Isso ocorre devido ao relacionamento identificador de cardinalidade `1:n` entre cada tabela *dimensão* e a tabela *fato*. A coluna `QTD` representa a métrica obtida pela combinação das dimensões.

Assim, por meio da restrição de integridade referencial, a coluna da tabela *fato* `PK_D01` referencia a coluna `PK_Dim-01` da `Dimensão-01`; a coluna `PK_D02` referencia a coluna `PK_Dim-02` da `Dimensão-02`; a

coluna `PK_D03` referencia a coluna `PK_Dim-03` da `Dimensão-03` e a coluna `PK_D04` referencia a coluna `PK_Dim-04` da `Dimensão-04`. Então, considerando uma linha da tabela *fato* cuja coluna `QTD` possua um valor, podemos inferir, por meio dessas referências, quais linhas das dimensões estão sendo combinadas para formar essa linha da tabela *fato*.

O modelo dimensional também pode ser representado por um cubo quando implementado em bancos de dados multidimensionais, no caso de existir até três dimensões, e por um hipercubo (um cubo com cubos no interior), quando existirem mais de três dimensões. A figura 7.4 apresenta o cubo de um modelo dimensional com três dimensões:

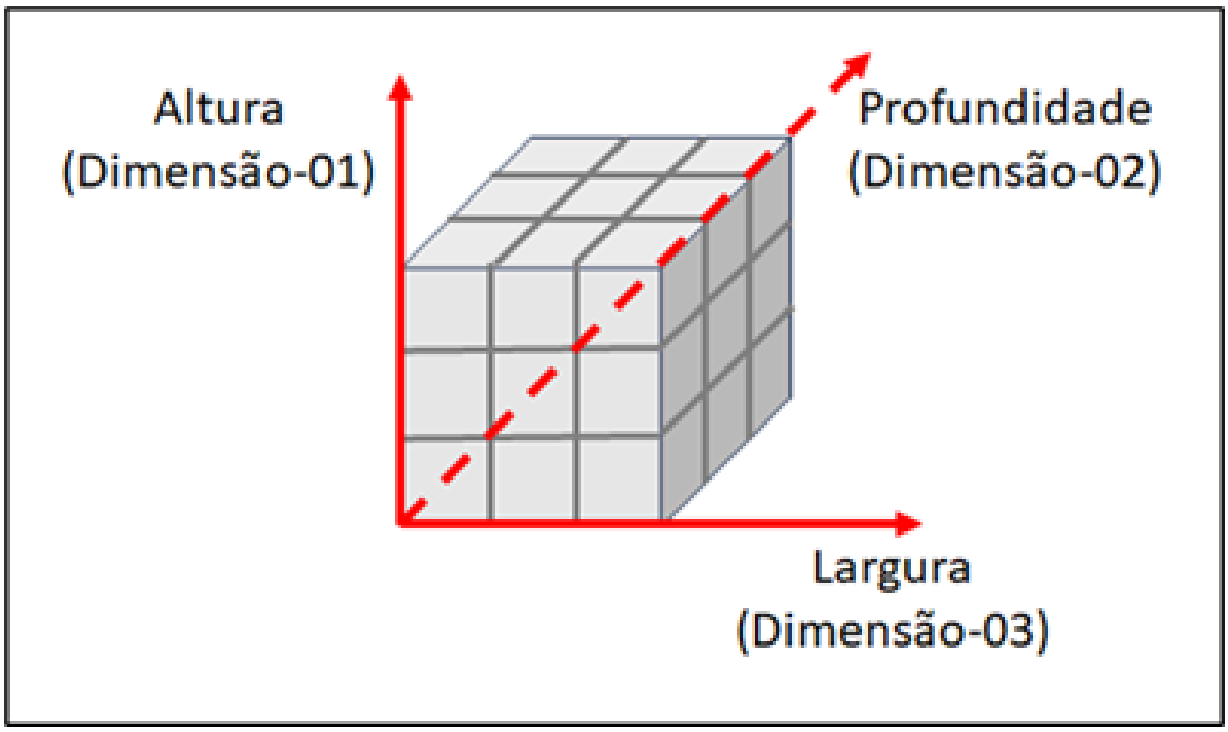


Figura 7.4: Representação do modelo dimensional com três dimensões por meio de um cubo.

Na figura 7.4, é possível observar que as três dimensões do cubo, a altura, a largura e a profundidade, possuem a mesma função das

dimensões do modelo estrela. Nesse modelo, o fato é armazenado na célula do cubo que representa a interseção das três dimensões.

De maneira geral, o processo de modelagem dimensional para a criação de um ambiente de DW pode ser resumido em quatro passos (KIMBALL, 2008):

1. **Escolher o processo de negócio:** priorizar os processos mais importantes da organização;
2. **Declarar o grão:** deve-se escolher o menor grão (maior nível de detalhamento) possível. Cada tabela *fato* deve ter um único grão;
3. **Identificar as dimensões:** a partir da escolha do grão, é possível escolher as dimensões compatíveis;
4. **Identificar os fatos:** escolher as métricas aplicáveis ao processo selecionado.

Para facilitar o entendimento do processo de modelagem dimensional, apresentaremos a seguir um exemplo de projeto para a criação de um ambiente de DW.

7.2 Exemplo de um projeto de modelagem dimensional de um ambiente de DW

Nesta seção, será apresentado um projeto simples de modelagem dimensional de um ambiente de DW seguindo as quatro etapas sugeridas por Kimball (2008), para que você compreenda as atividades e artefatos gerados em cada etapa da modelagem.

Escolher o processo de negócio

Para a implementação deste exemplo, foi escolhido o processo de negócio relacionado a vendas de uma empresa de varejo fictícia, denominada “Best Sale”, com filiais em vários estados brasileiros.

Serão considerados dados internos e externos a essa organização para a criação e carga do modelo dimensional. A figura 7.5 apresenta o fluxo das informações no ambiente de DW a ser projetado:

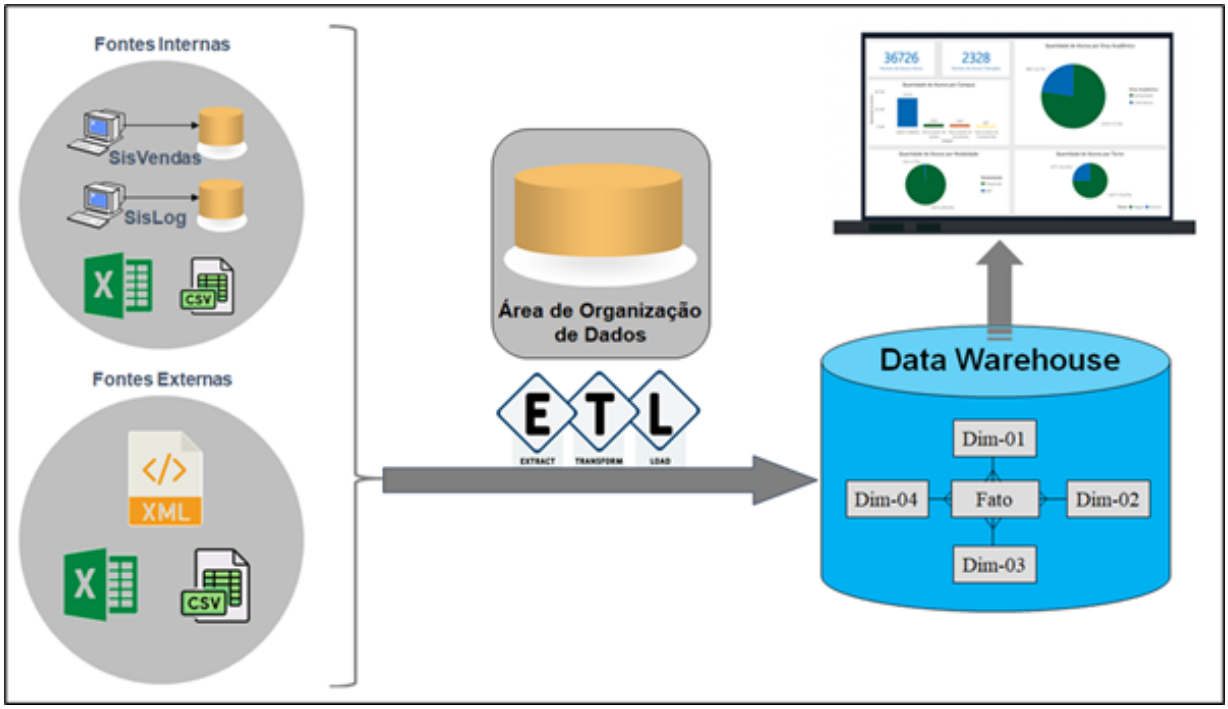


Figura 7.5: Ambiente de DW da empresa Best Sale.

O ambiente de DW apresentado na figura 7.5 é composto por quatro camadas. A primeira é formada pelas fontes de dados internas e externas à empresa. As fontes internas são formadas pelas bases de dados de dois sistemas transacionais: o SisVendas, que automatiza os processos envolvidos na venda de produtos, e o SisLog, que é um sistema de logística utilizado para automatizar os processos relacionados à entrega dos produtos vendidos. Há também planilhas Excel e arquivos no formato CSV com informações sobre as vendas que são gerados internamente à empresa. As fontes externas são formadas por arquivos XML, CSV e planilhas Excel obtidos em portais de dados abertos do governo brasileiro.

A segunda é a camada ETL (*Extraction*, *Transfomation*, *Load*), responsável pela extração, transformação e carga dos dados. Para a

realização dessas atividades, normalmente é criada uma área de organização de dados, onde são armazenados os dados a serem transformados, os programas que implementam essas transformações e os metadados que descrevem esses dados e programas. Nessa camada, os dados são extraídos das fontes internas e externas, passam por vários processos de transformação para depois serem carregados no DW. Como exemplos de transformações dos dados, podemos citar a uniformização da representação do sexo dos clientes e dos preços de catálogo dos produtos. O sexo pode ser registrado em várias fontes de formas diferentes: uma fonte pode possuir os valores "H" para *homens* e "M" para *mulheres*, entre outros; outra fonte pode representar *masculino* com "M" e *feminino* com "F". Os preços dos produtos podem estar em moedas distintas: em uma fonte, pode estar em Reais e, em outra, em Euros ou Dólares, por exemplo. Antes de esses dados serem carregados no DW, eles precisam ser uniformizados, ou seja, é necessário escolher uma representação padrão e transformar todas as outras representações naquela escolhida.

A terceira camada consiste em um repositório de dados dimensional, o DW, local de armazenamento dos dados. Finalmente, a quarta camada é responsável pelo processamento dos dados armazenados no DW para, posteriormente, montar e apresentar os relatórios sumarizados e *dashboards* (gráficos) para os usuários do ambiente.

Declarar o grão

O grão escolhido será uma linha de uma nota fiscal emitida pela empresa ao vender um ou mais produtos. Normalmente, uma linha de uma nota fiscal possui o número de ordem, o código do produto vendido, a descrição, o preço unitário, a quantidade de um mesmo produto comprado e o valor correspondente à quantidade do item multiplicado pelo preço unitário. A figura 7.6 apresenta um exemplo de nota fiscal da empresa Best Sale:

Empresa Best Sale					
Nº da Venda:	1	Data da Venda:	05/12/2022		
CPF Cliente:	111	Nome Cliente:	Ana Maria		
Matrícula Vendedor:	852	Nome Vendedor:	José Silva		

Nr Ordem	Código	Produto	Preço	Qtd	Total
1	2222	Impressora	450,00	3	1.350,00
2	3333	Fogão	650,32	2	1.300,64
3	4444	Televisão	810,65	1	810,65
Total					R$ 3.461,29

Figura 7.6: Nota fiscal da empresa Best Sale.

Como pode ser observado na figura 7.6, o primeiro item da nota fiscal possui o número de ordem igual a 1 e o código 2222; trata-se de uma impressora cujo preço unitário é R$ 450,00. Foram vendidas 3 impressoras a um valor total de R$ 1.350,00.

Identificar as dimensões

As dimensões descrevem as características relevantes que comporão o fato. Poderíamos escolher várias características para descrever o fato `vendas` da empresa Best Sale, porém, para simplificação do modelo, consideraremos apenas as dimensões `Tempo`, `Geografia`, `Produto` e `Cliente`.

A dimensão `Tempo` existe em todos os modelos dimensionais. No modelo a ser desenvolvido, ela representará a data da venda. A dimensão `Geografia` representará a localização da loja que emitiu a nota fiscal. A dimensão `Produto` descreverá o item comprado. Finalmente, a dimensão `Cliente` descreverá a pessoa que comprou os produtos. As dimensões são apresentadas nas figuras 7.7, 7.8, 7.9 e 7.10.

Dim_Tempo

pk_tempo	data	ano	mes	dia	trimestre	semestre	nome_dia	nome_mes
1	2022-01-01	2022	1	1	1	1	Sábado	Janeiro
2	2022-01-02	2022	1	2	1	1	Domingo	Janeiro
3	2022-01-03	2022	1	3	1	1	Segunda	Janeiro
4	2022-01-04	2022	1	4	1	1	Terça	Janeiro
5	2022-01-05	2022	1	5	1	1	Quarta	Janeiro

Figura 7.7: Dimensão Tempo.

As colunas da tabela de dimensão `Tempo` apresentada na figura 7.7 são `pk_tempo`, `data`, `ano`, `mes`, `dia`, `trimestre`, `semestre`, `nome_dia` e `nome_mes`. A coluna `pk_tempo` é a chave primária da tabela. Essa chave primária é artificial, ou seja, é uma sequência de números que foi criada artificialmente para diferenciar um registro dos demais; por isso, ela é denominada **chave substituta** (*surrogate key*). Em projetos dimensionais, toda tabela de dimensão possui uma chave primária artificial formada por apenas uma coluna (normalmente uma sequência de números), para possibilitar simplicidade, melhor performance das consultas analíticas e o versionamento dos dados.

O campo `data` representa a data da compra do produto: os campos `ano`, `mes`, `dia`, `trimestre`, `semestre`, `nome_dia` e `nome_mes` poderiam ser obtidos facilmente no momento da consulta a partir da data armazenada; porém, eles são armazenados, caracterizando a redundância dos dados, para agilizar a geração dos relatórios analíticos.

Dim_Geografia

pk_geografia	sigla_uf	nome_uf	sigla_regiao	nome_regiao
1	RO	Rondônia	N	Norte
2	AC	Acre	N	Norte
3	AM	Amazonas	N	Norte
4	RR	Roraima	N	Norte
5	PA	Pará	N	Norte

Figura 7.8: Dimensão Geografia.

As colunas da tabela de dimensão `Geografia` apresentada na figura 7.8 são `pk_geografia`, `sigla_uf`, `nome_uf`, `sigla_regiao` e `nome_regiao`. A coluna `pk_geografia` é a chave primária artificial da tabela. As outras colunas representam a sigla da UF, o nome da UF, a sigla da região geográfica onde a UF está localizada e o nome dessa região.

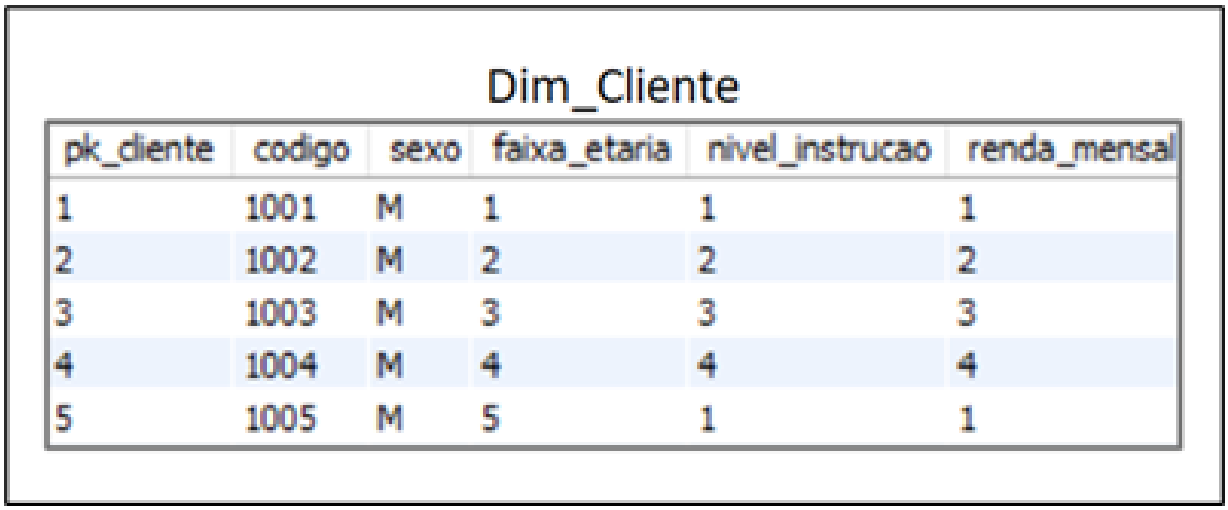
Dim_Produto

pk_produto	nome	marca	categoria	preco
1	PC	Apple	Informática	5000.00
2	PC	Dell	Informática	3500.00
3	PC	HP	Informática	4000.00
4	Notebook	Apple	Informática	8000.00
5	Notebook	Dell	Informática	5000.00

Figura 7.9: Dimensão Produto.

As colunas da tabela de dimensão `Produto` apresentada na figura 7.9 são `pk_produto`, `nome`, `marca`, `categoria` e `preço`. A coluna `pk_produto`

é a chave primária artificial da tabela. Os outros atributos representam o nome, a marca, a categoria e o preço do produto.



Dim_Cliente

pk_diente	codigo	sexo	faixa_etaria	nivel_instrucao	renda_mensal
1	1001	M	1	1	1
2	1002	M	2	2	2
3	1003	M	3	3	3
4	1004	M	4	4	4
5	1005	M	5	1	1

Figura 7.10: Dimensão Cliente.

As colunas da tabela de dimensão `Cliente` apresentada na figura 7.10 são `pk_cliente`, `codigo`, `sexo`, `faixa_etaria`, `nivel_instrução` e `renda_mensal`. A coluna `pk_cliente` é a chave primária artificial da tabela. Os outros atributos representam o código do cliente, o sexo, a faixa etária, o nível de instrução e a renda mensal. As faixas etárias são: 1, para pessoas com idade entre 18 e 25 anos; 2, para pessoas com idade entre 26 e 35 anos; 3, para pessoas com idade entre 36 e 45 anos; 4, para pessoas com idade entre 46 e 55 anos; 5, para pessoas com idade entre 56 e 65 anos; e 6, para pessoas com idade superior a 65 anos. Os níveis de instrução são: 1, para pessoas que estudaram até o 2º grau incompleto; 2, para o 2º grau completo; 3, para graduação; e 4, para pós-graduação. As rendas mensais são: 1, para pessoas que ganham até 5 salários mínimos; 2, para 6 até 10 salários mínimos; 3, para 11 até 20 salários mínimos; e 4, para acima de 20 salários mínimos.

Identificar os fatos

O modelo a ser desenvolvido possuirá três fatos: o primeiro será o preço unitário do produto vendido, o segundo será a quantidade de

produtos vendidos e o último será o valor unitário do produto multiplicado pela quantidade vendida. A figura 7.11 apresenta uma representação do modelo estrela do processo de vendas da empresa Best Sale:

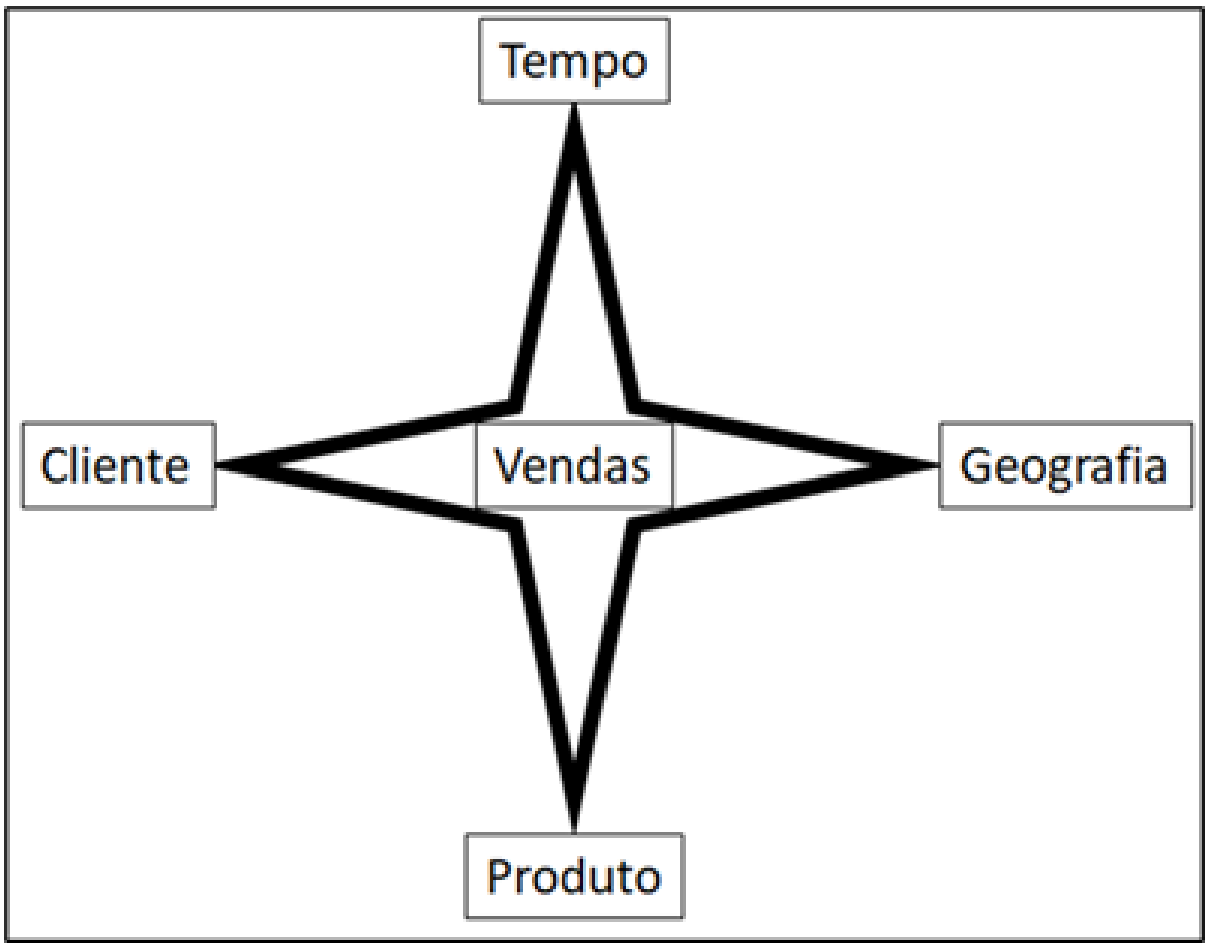


Figura 7.11: Modelo estrela do processo de vendas da empresa Best Sale.

O modelo estrela apresentado na figura 7.11 possui as dimensões `Tempo`, `Geografia`, `Produto` e `Cliente`, e o fato `Vendas`. As dimensões descrevem as características do negócio, enquanto o fato quantifica; ou seja, nesse caso o fato (fato `Vendas`) representa a quantidade de vendas realizadas em uma data (dimensão `Tempo`), por uma loja (dimensão `Geografia`), de um produto (dimensão `Produto`) a um cliente (dimensão `Cliente`).

Esse modelo estrela, no nível conceitual, pode ser representado por meio do modelo entidade-relacionamento utilizando a nomenclatura da engenharia da informação, conforme é apresentado na figura 7.12:

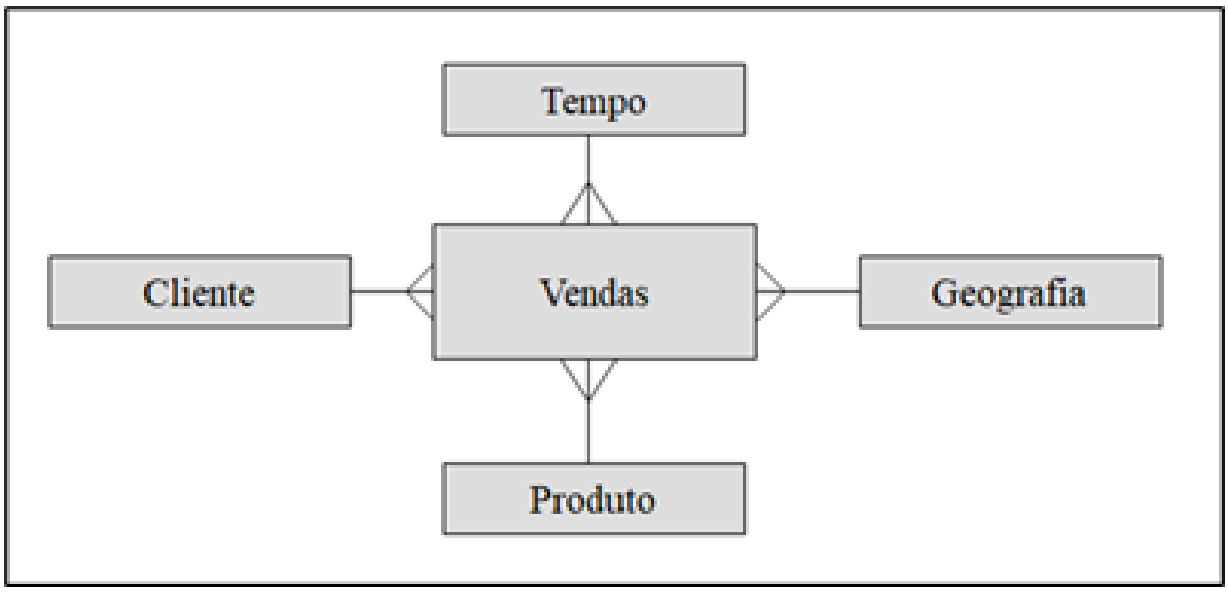


Figura 7.12: Representação do modelo estrela do processo de vendas da empresa Best Sale por meio do modelo entidade-relacionamento.

O modelo entidade-relacionamento representado na figura 7.12 é composto por cinco entidades. A entidade central representa o fato `Vendas` e as periféricas são as dimensões. É possível observar que existe um relacionamento identificador de cardinalidade `1:n` entre cada dimensão e o fato. Assim, quando essas entidades forem mapeadas para tabelas, a tabela *fato* será composta por, no mínimo, quatro colunas, e cada uma delas será uma chave estrangeira que referenciará a chave primária de cada dimensão. O conjunto dessas quatro chaves estrangeiras comporá a chave primária da tabela *fato* `Vendas` . Por isso, a tabela *fato* é considerada uma variável dependente, pois depende das dimensões para existir, e as dimensões são consideradas variáveis independentes.

A principal diferença entre o modelo estrela e o modelo flocos de neve é a normalização das dimensões. O modelo flocos de neve é

equivalente ao modelo estrela com as dimensões normalizadas até a 3FN (Terceira Forma Normal).

É possível observar que a dimensão `Geografia`, representada pela tabela `Dim_Geografia` apresentada na figura 7.8, não está na 3FN, pois basta saber a UF para determinar a `Região`. Isso caracteriza dependência transitiva, impedindo que essa tabela atinja a 3FN.

Em modelos dimensionais do tipo estrela, é muito comum as tabelas de dimensões conterem dados redundantes e não respeitarem a 3FN para facilitarem a construção e a execução de consultas analíticas aos dados. Por outro lado, nos modelos flocos de neve, essas dimensões são normalizadas para respeitarem a 3FN. Então, pode-se inferir que o modelo flocos de neve é equivalente ao modelo estrela com as suas dimensões normalizadas até a 3FN.

A figura 7.13 apresenta o modelo flocos de neve equivalente ao modelo estrela apresentado na figura 7.12:

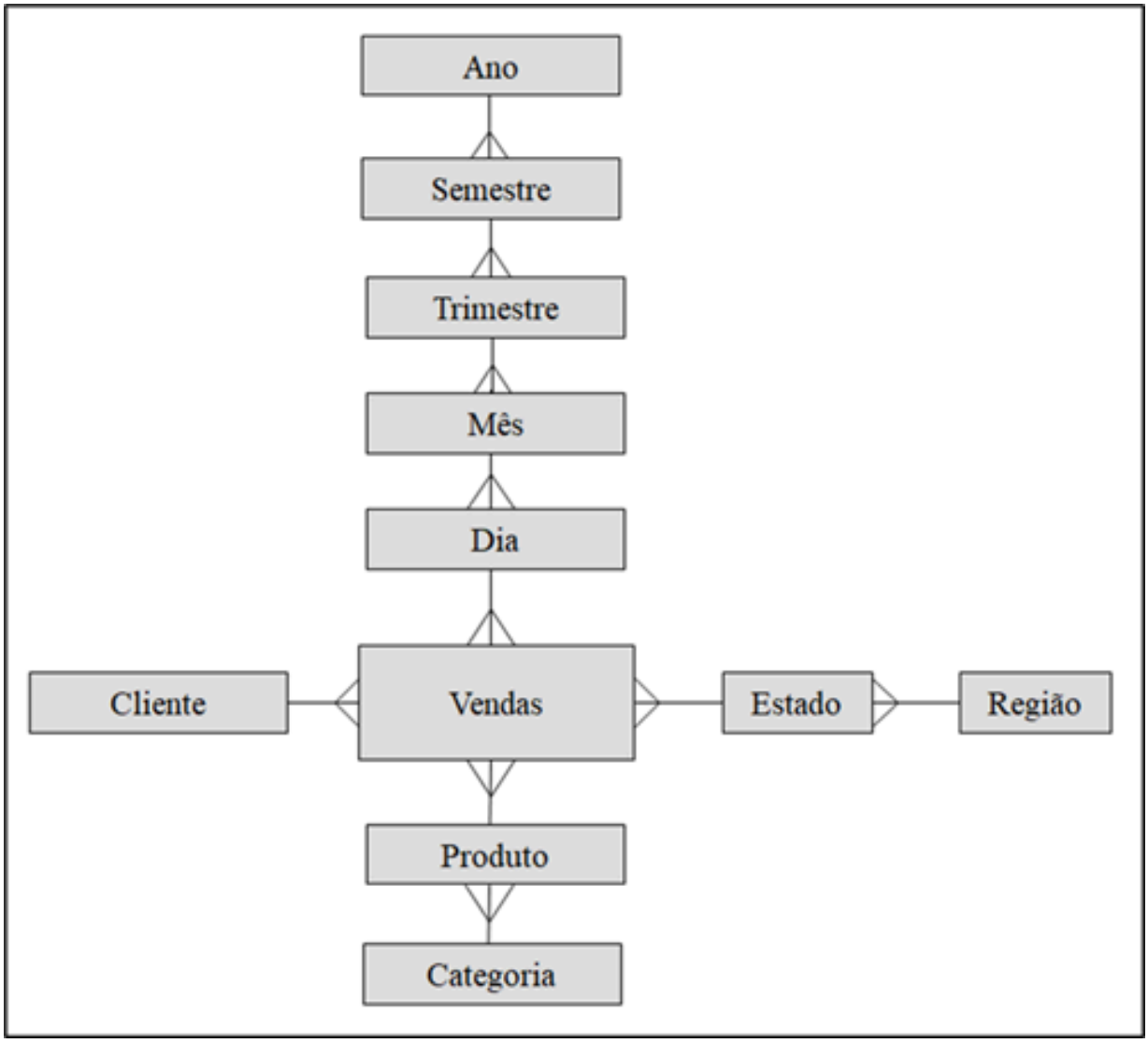


Figura 7.13: Representação do modelo flocos de neve do processo de vendas da empresa Best Sale por meio do modelo entidade-relacionamento.

O modelo entidade-relacionamento representado na figura 7.13 é composto por 11 entidades. A entidade central representa o fato `Vendas` e as periféricas são as dimensões. É possível observar que as dimensões `Tempo`, `Geografia` e `Produto` da figura 7.12 foram normalizadas até a 3FN, dando origem a novas entidades representadas na figura 7.13.

A dimensão `Tempo`, representada pela tabela `Dim_Tempo` apresentada na figura 7.7, foi normalizada e mapeada para as entidades Dia, Mês,

Trimestre, Semestre e Ano. A dimensão Geografia, representada pela tabela Dim_Geografia apresentada na figura 7.8, foi mapeada para as entidades Estado e Região. Finalmente, a entidade Produto, representada pela tabela Dim_Produto apresentada na figura 7.9, foi mapeada para as entidades Produto e Categoria.

Existe um relacionamento identificador de cardinalidade 1:n entre as *dimensões* Dia, Estado, Produto e Cliente e o *fato* Vendas. Assim, quando essas entidades forem mapeadas para tabelas, a tabela fato será composta por, no mínimo, quatro colunas, e cada uma delas será uma chave estrangeira que referenciará a chave primária de cada uma dessas dimensões. O conjunto dessas quatro chaves estrangeiras comporá a chave primária da tabela *fato* Vendas. Essa estrutura é similar à estrutura do modelo estrela.

No modelo flocos de neve, para obter todos os dados da dimensão Tempo do modelo estrela, é necessário fazer uma junção interna entre as tabelas Dia, Mês, Trimestre, Semestre e Ano. Isso se deve à normalização dessa dimensão no modelo flocos de neve. Situação semelhante acontece com as dimensões Geografia e Produto. Dependendo do volume de dados dessas tabelas, a operação de junção interna pode ser demorada e prejudicar a performance das consultas; por isso, caso não haja uma justificativa para se usar o modelo flocos de neve, normalmente a escolha do modelo estrela é a mais indicada.

As entidades apresentadas na figura 7.12, que representam o modelo estrela no nível conceitual, quando mapeadas para o nível lógico, gerarão as tabelas apresentadas na figura 7.14:

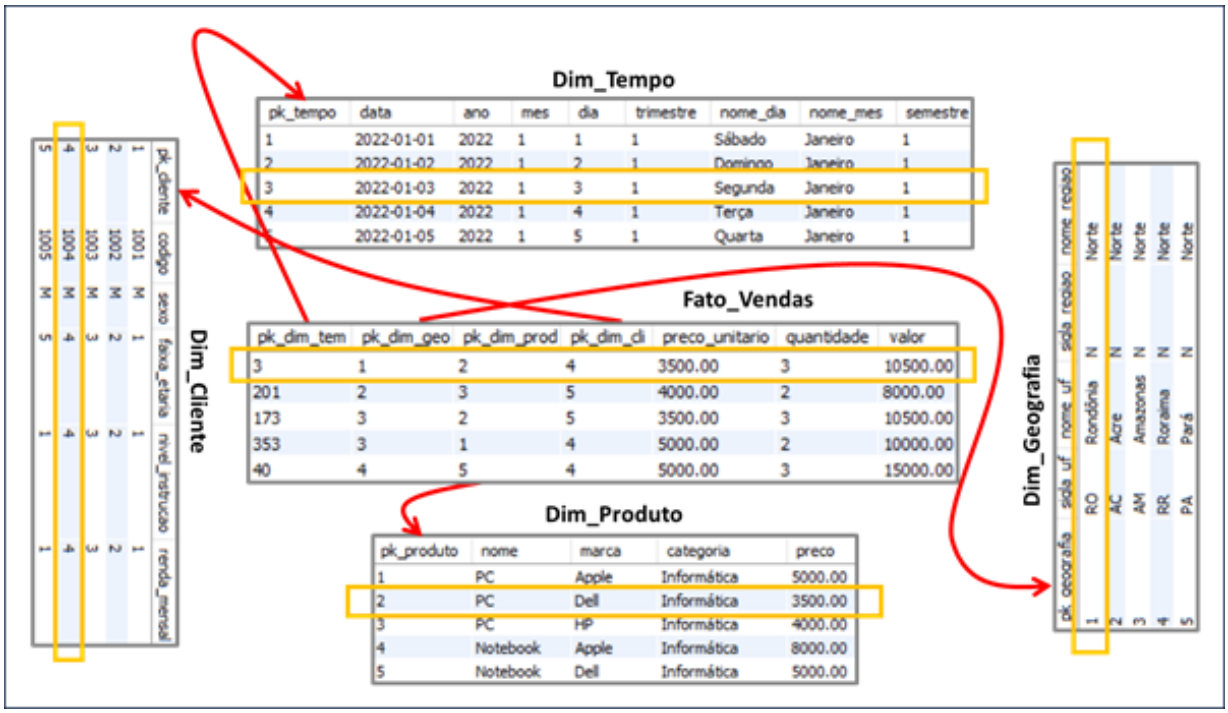


Figura 7.14: Modelo lógico dimensional relativo a vendas da empresa Best Sale.

Na figura 7.14, é possível observar a tabela `Fato_Vendas`, na posição central do modelo, e as tabelas dimensões `Dim_Tempo`, `Dim_Geografia`, `Dim_Produto` e `Dim_Cliente`. Nessa figura, as tabelas `Dim_Geografia` e `Dim_Cliente` estão dispostas na vertical.

Como toda tabela *fato*, a tabela `Fato_Vendas` possui uma chave primária composta pelas chaves estrangeiras que referenciam as chaves primárias das tabelas *dimensões*. Assim, a chave primária da tabela `Fato_Vendas` é composta pelas colunas `pk_dim_tem`, `pk_dim_geo`, `pk_dim_prod` e `pk_dim_cli`.

A coluna `pk_dim_tem` referencia a coluna `pk_tempo` da tabela `Dim_Tempo`; a coluna `pk_dim_geo` referencia a coluna `pk_geografia` da tabela `Dim_Geografia`; a coluna `pk_dim_prod` referencia a coluna `pk_produto` da tabela `Dim_Produto` e a coluna `pk_dim_cli` referencia a coluna `pk_cliente` da tabela `Dim_Cliente`. As setas da figura 7.14 representam essas referências.

Considerando a linha da tabela `Fato_Vendas` destacada pelo retângulo — cujo conteúdo de cada coluna é: `pk_dim_tem = 3` ; `pk_dim_geo = 1` ; `pk_dim_prod = 2` ; `pk_dim_cli = 4` ; `preco_unitario = 3.500,00` , `quantidade = 3` e `valor = 10500.00` —, é possível observar que:

- `Dim_Tempo` (coluna `pk_tempo = 3`): representa a data `2022-01-03` (ano-mês-dia), ou seja, em 3 de janeiro de 2022, uma segunda-feira, primeiro trimestre e primeiro semestre do ano;
- `Dim_Geografia` (coluna `pk_geografia = 1`): representa a loja da rede Best Sale localizada em Roraima, na região Norte do Brasil;
- `Dim_Produto` (coluna `pk_produto = 2`): representa o PC Dell, um item de informática, que custa R$ 3.500,00;
- `Dim_Cliente` (coluna `pk_cliente = 4`): representa o cliente cujo código é 1004, do sexo masculino, que pertence à faixa etária de 46 até 55 anos, possui pós-graduação e possui uma renda superior à 20 salários mínimos;
- Fato `Vendas` (colunas `preco_unitario` , `quantidade` e `valor`): significa o valor unitário de R$ 3.500,00 de 3 produtos vendidos, totalizando R$ 10.500,00.

Sendo assim, pode-se inferir que as linhas da tabela `Fato_Vendas` e das dimensões `Dim_Tempo` , `Dim_Geografia` , `Dim_Produto` e `Dim_Cliente` destacadas na figura 7.14 significam que:

Na data 2022-01-03 (ano-mês-dia), ou seja, em 3 de janeiro de 2022, uma segunda-feira, primeiro trimestre e primeiro semestre do ano, na loja da rede Best Sale localizada em Roraima, na região Norte do Brasil, o cliente cujo código é 1004, do sexo masculino, pertence à faixa etária de 46 até 55 anos, possui pós-graduação e uma renda superior a 20 salários-mínimos, comprou 3 PCs Dell, um item de informática, que custa R$ 3.500,00, totalizando o valor de R$ 10.500,00.

A figura 7.15 apresenta essa venda de forma simplificada:

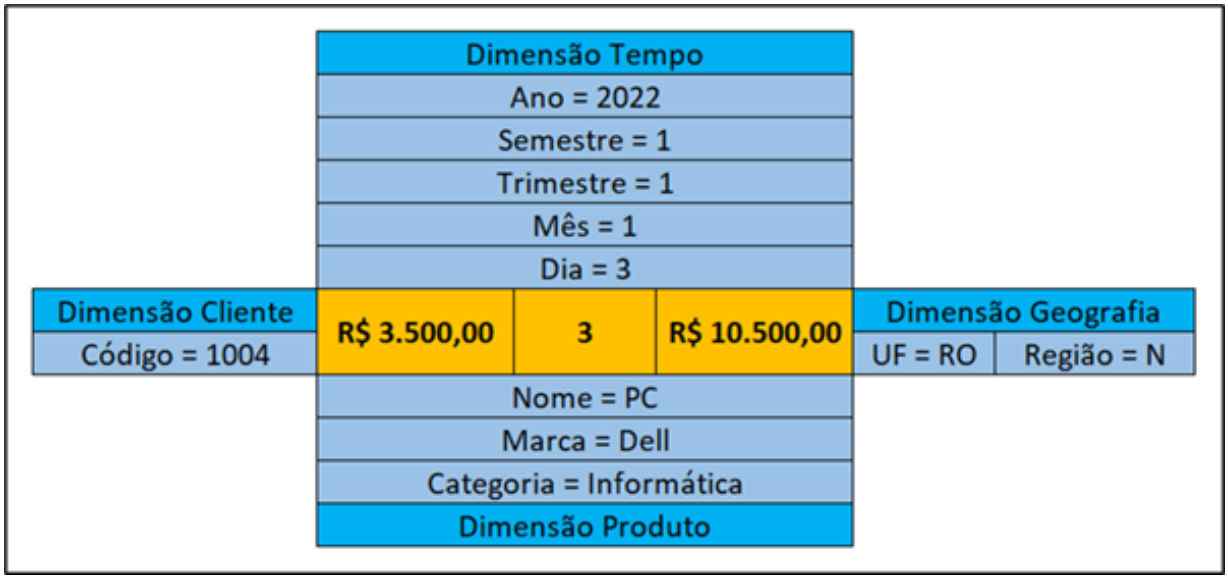


Figura 7.15: Representação de uma venda da empresa Best Sale.

Cabe ressaltar que o modelo dimensional também pode ser representado por meio de um cubo ou hipercubo. Para representar o modelo dimensional do fato `Vendas` da empresa Best Sale por meio do cubo, foi retirada a dimensão `Cliente` do modelo apresentado na figura 7.11 com o objetivo de tornar o modelo tridimensional. A figura 7.16 apresenta esse novo modelo:

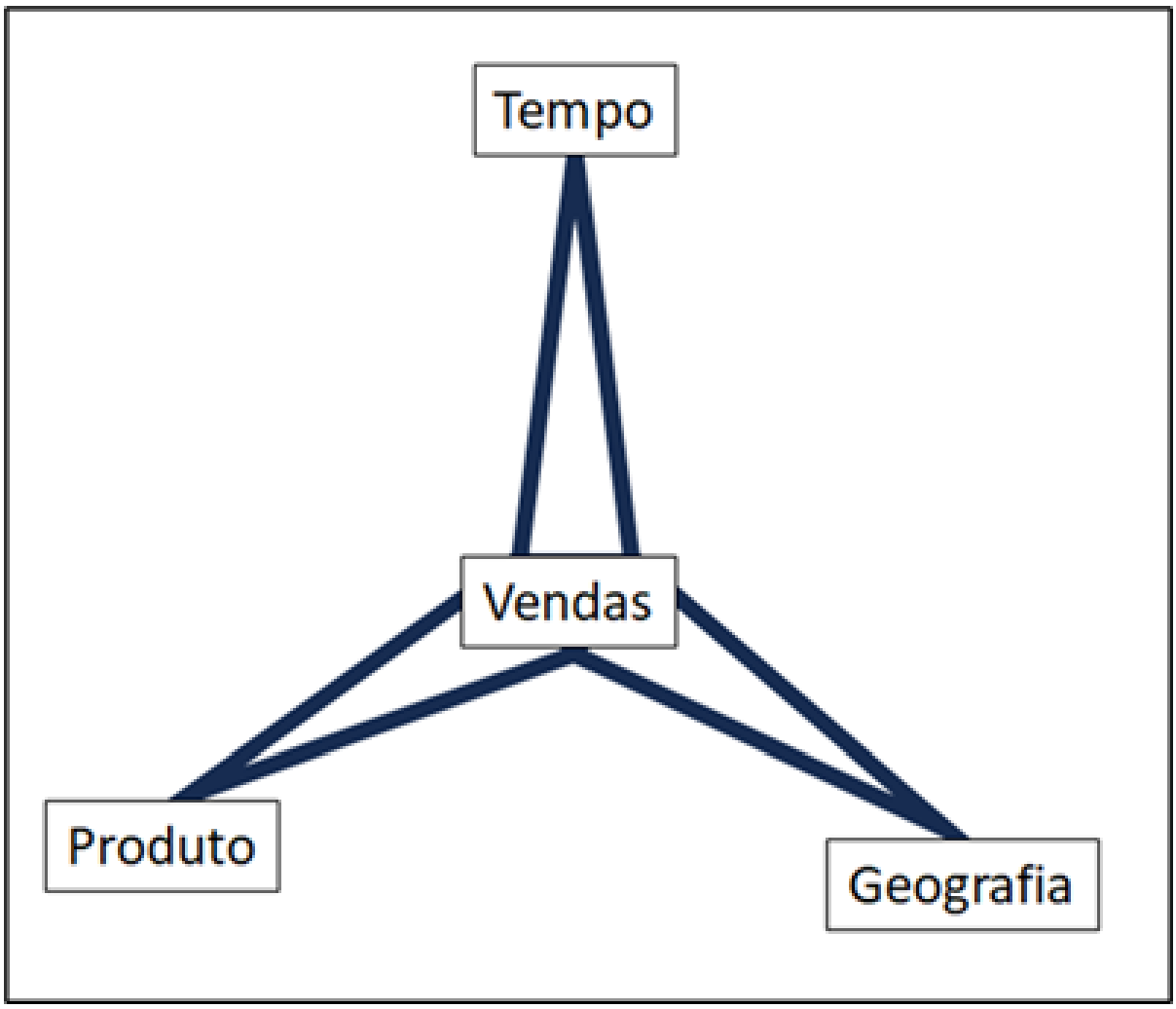


Figura 7.16: Modelo estrela com três dimensões do processo de vendas da empresa Best Sale.

O modelo apresentado na figura 7.16 apresenta o fato `Vendas` e as dimensões `Tempo`, `Geografia` e `Produto`. Esse modelo pode ser mapeado para o cubo da figura 7.17:

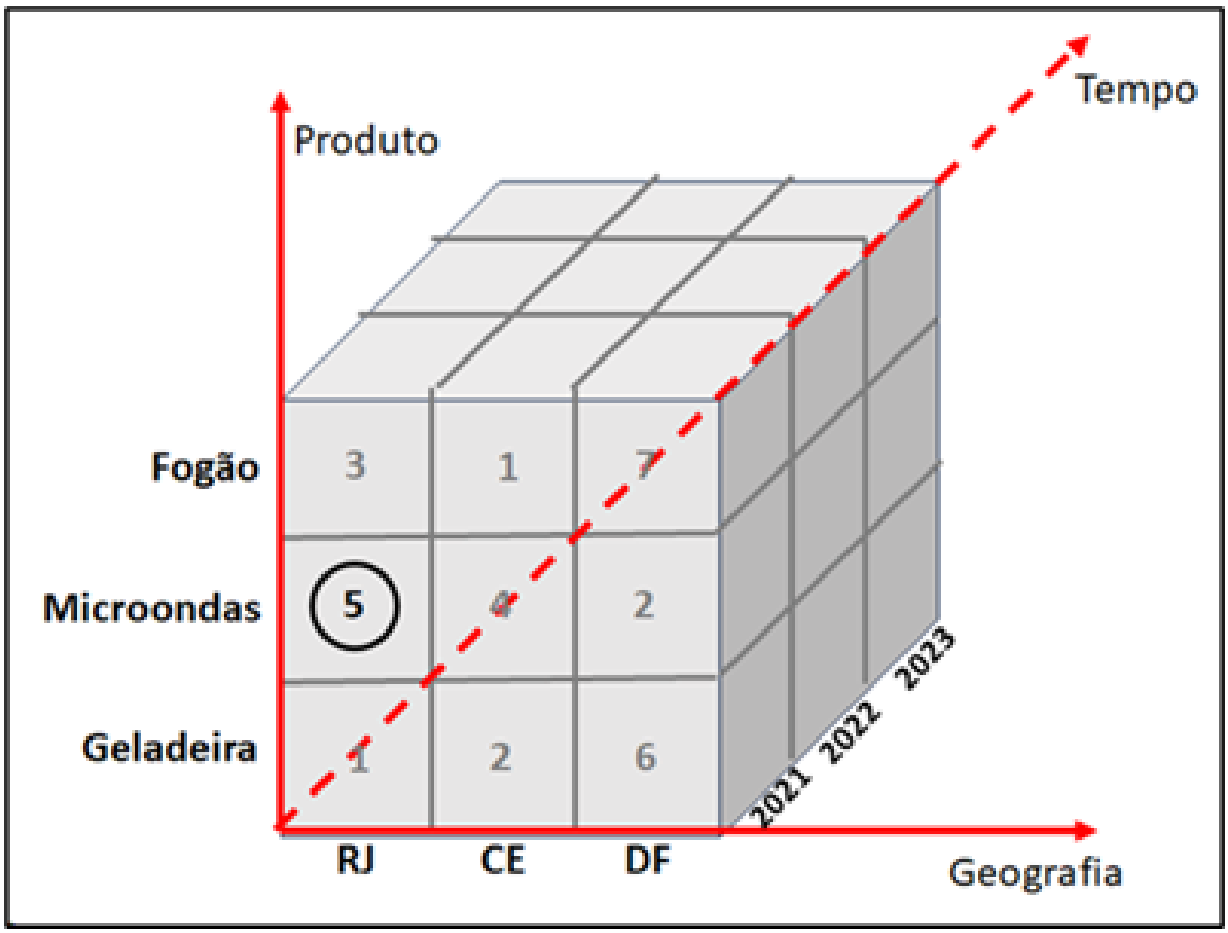


Figura 7.17: O cubo relativo a vendas da empresa Best Sale.

Na figura 7.17, os três eixos representam as dimensões e as células do cubo representam os fatos armazenados. Então, a célula do cubo com o valor `5` realçada por um círculo representa a venda de 5 micro-ondas (dimensão `Produto`) no ano de 2021 (dimensão `Tempo`) no estado do Rio de Janeiro (dimensão `Geografia`). Cabe ressaltar que os usuários visualizam o cubo (ou seja, as dimensões e os fatos) no formato de relatório. A figura 7.18 apresenta o relatório visualizado pelos usuários com os dados do cubo da figura 7.17:

	2021			2022			2023		
	RJ	CE	DF	RJ	CE	DF	RJ	CE	DF
Fogão	3	1	7	2	4	2	3	7	9
Microondas	5	4	2	4	9	7	5	6	1
Geladeira	1	2	6	2	1	4	5	7	6

Figura 7.18: Relatório gerado a partir do cubo relativo às vendas da empresa Best Sale.

No relatório da figura 7.18, é possível observar os anos 2021, 2022 e 2023 da dimensão `Tempo`; as UFs RJ, CE e DF da dimensão `Geografia`; os produtos Fogão, Micro-ondas e Geladeira da dimensão `Produto`; e as quantidades vendidas de cada produto em uma determinada UF em um ano específico, que representam o fato `Vendas`. Destaca-se que a visualização da figura 7.17 fica limitada ao ano de 2021, porém no relatório é possível visualizar as vendas de todos os anos.

7.3 Consultas OLAP

Conforme apresentado no capítulo 3, a diferença entre a arquitetura OLTP, dos sistemas transacionais, e a arquitetura OLAP, dos sistemas de apoio à decisão, é que a primeira arquitetura visa dar suporte ao fluxo dos dados gerados e consumidos pelos processos operacionais das organizações — por exemplo, gerar notas fiscais das empresas, cadastrar notas de alunos em escolas, cadastrar resultados de exames em laboratórios, entre outros.

Assim, na arquitetura OLTP, os dados podem ser inseridos, consultados, atualizados e excluídos dos repositórios. Uma das tecnologias mais utilizadas nessa arquitetura para o armazenamento

de dados são os bancos de dados relacionais apresentados no capítulo 4.

Por outro lado, na arquitetura OLAP os dados são carregados periodicamente, de acordo com as necessidades do negócio, em repositórios dimensionais denominados *Data Warehouse*, para serem utilizados para subsidiar decisões. As consultas OLAP são caracterizadas pelo grande volume de dados manipulado, por utilizarem relatórios sumarizados e gráficos (*dashboards*) para facilitar o entendimento dos dados e pela flexibilidade em reorganizar os dados para a geração de novas visualizações.

As consultas OLAP mais comuns são:

- ***Slice-and-dice* (fatiar e cortar):** realizar a operação de projeção nas dimensões;
- ***Pivot* (pivoteamento):** rotação do cubo ou hipercubo;
- ***Roll-up* ou *drill-up* (agregação)**: os dados são resumidos com generalização crescente (dia, mês, ano);
- ***Drill-down* (desmembramento):** níveis crescentes de detalhes são revelados (ano, mês, dia).

Slice-and-Dice

A consulta OLAP *slice-and-dice* realiza a operação de projeção nas dimensões. Essa operação pode ser interpretada como fatiar e cortar o modelo dimensional em cubos ou fatias.

A operação ***slice*** seleciona dados de uma única dimensão de um cubo OLAP. As figuras 7.19 e 7.20 apresentam um exemplo da consulta *slice*.

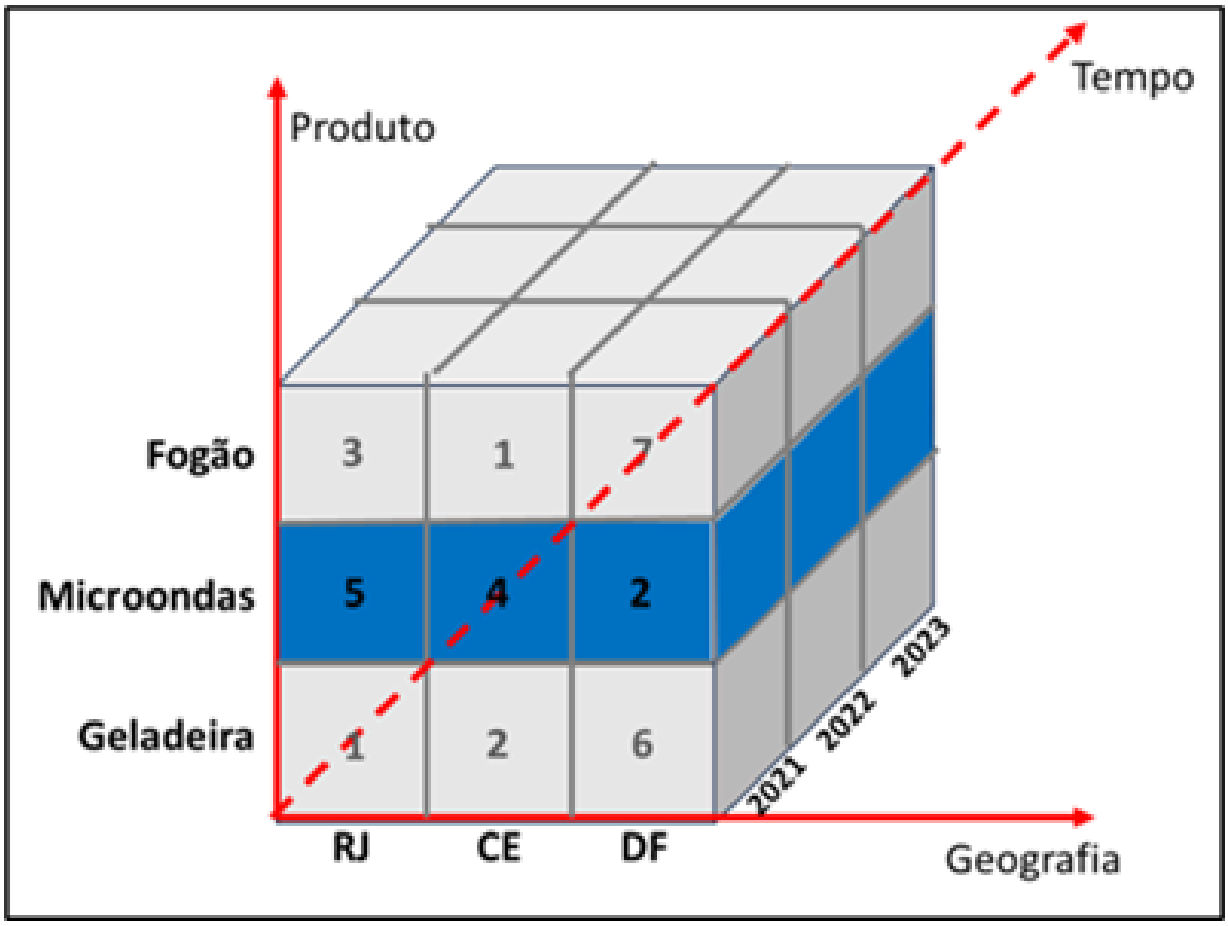


Figura 7.19: Consulta slice ao cubo relativo a vendas da empresa Best Sale.

Como pode ser observado na linha realçada da figura 7.19, o cubo relativo a vendas da empresa Best Sale foi fatiado para apresentar as vendas de apenas um produto, o micro-ondas. Assim, é possível observar na fatia realçada as vendas de micro-ondas em todos os estados e anos. A parte inferior da figura 7.20 apresenta o relatório visualizado pelo usuário relativo à consulta *slice* apresentada na figura 7.19:

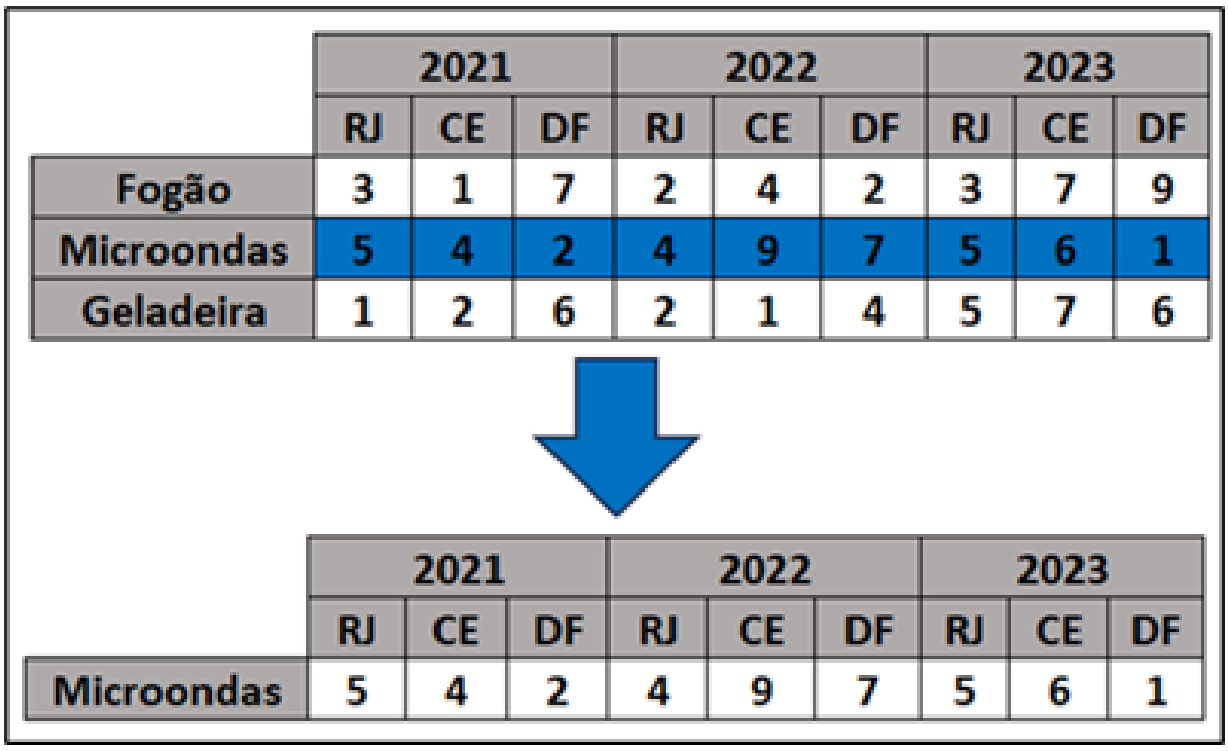


Figura 7.20: Relatório relativo à consulta slice.

A operação ***dice*** extrai um subcubo do cubo original executando uma operação de seleção em duas ou mais dimensões. As figuras 7.21 e 7.22 apresentam um exemplo da consulta *dice*.

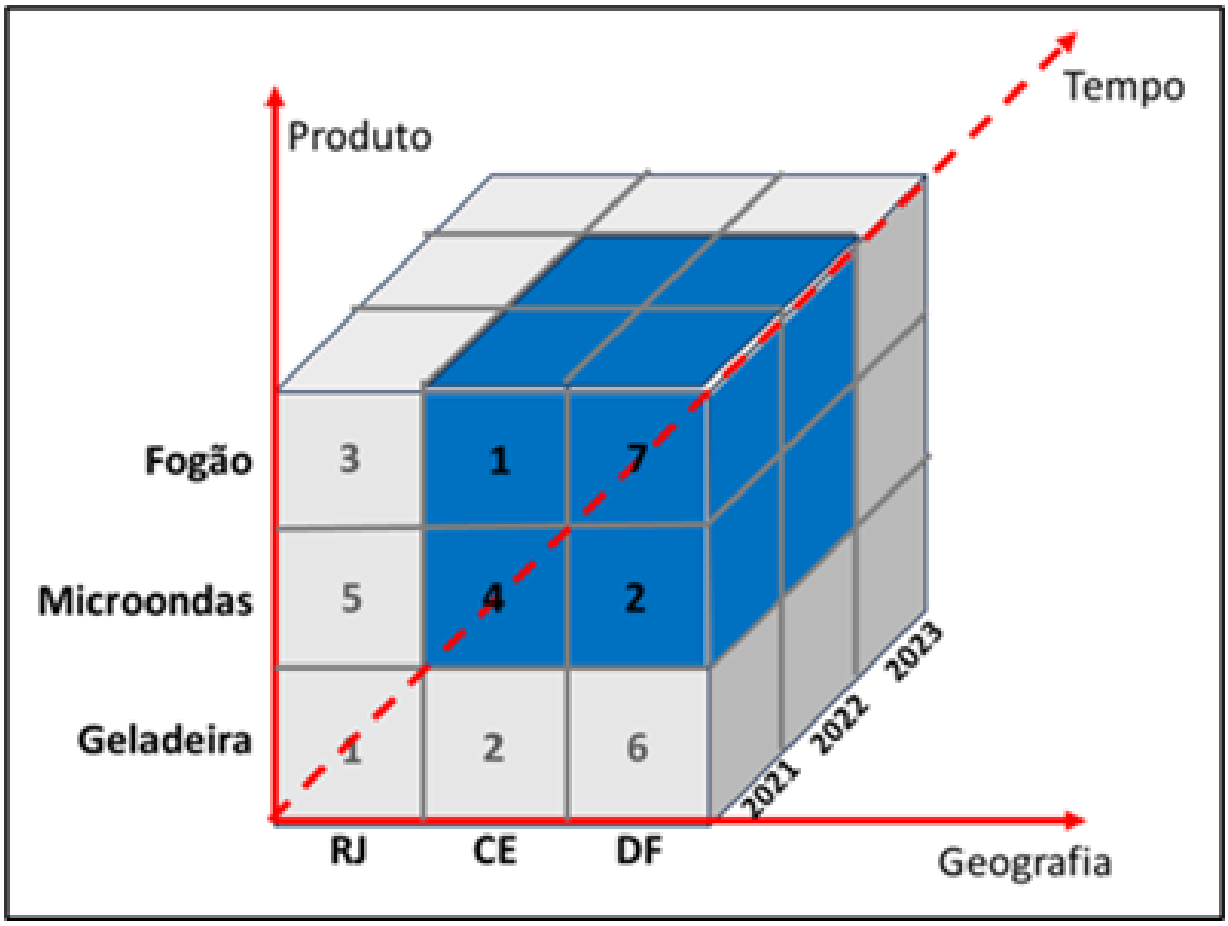


Figura 7.21: Consulta dice ao cubo relativo a vendas da empresa Best Sale.

A parte realçada da figura 7.21 apresenta o novo cubo (subcubo) resultante da consulta *dice* ao cubo relativo a vendas da empresa Best Sale. Esse novo cubo representa um subconjunto do cubo original. Assim, é possível observar no cubo realçado as vendas de fogão e micro-ondas, nos anos de 2021 e 2022, nas UFs CE e DF. A parte inferior da figura 7.22 apresenta o relatório visualizado pelo usuário relativo à consulta *dice* apresentada na figura 7.21.

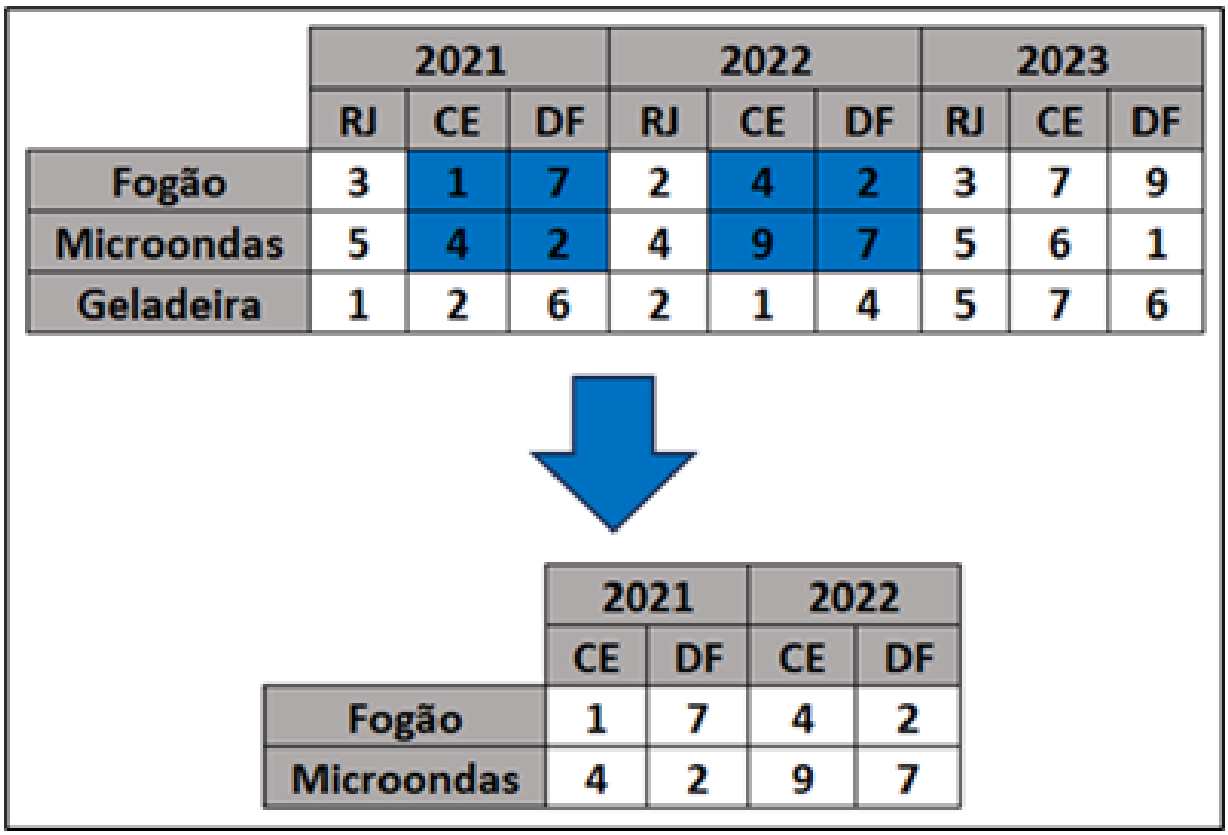


Figura 7.22: Relatório relativo à consulta dice.

Pivot

O pivoteamento consiste em "girar o cubo"; assim, as dimensões que compõem o relatório mudam de lugar. O objetivo é mudar o layout do relatório de acordo com o interesse do usuário. A figura 7.23 apresenta um exemplo de pivoteamento do cubo relativo a vendas da empresa Best Sale.

	2021			2022			2023		
	RJ	CE	DF	RJ	CE	DF	RJ	CE	DF
Fogão	3	1	7	2	4	2	3	7	9
Microondas	5	4	2	4	9	7	5	6	1
Geladeira	1	2	6	2	1	4	5	7	6

	2021			2022			2023		
	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira
RJ	3	5	1	2	4	2	3	5	5
CE	1	4	2	4	9	1	7	6	7
DF	7	2	6	2	7	4	9	1	6

Figura 7.23: Relatório relativo ao pivoteamento.

Na parte superior da figura 7.23, as dimensões `Tempo` e `Geografia` estão na horizontal e a dimensão `Produto` está na vertical do relatório. Após o pivoteamento, conforme a parte inferior dessa figura, as dimensões `Produto` e `Geografia` trocaram de lugar. Assim, as dimensões `Tempo` e `Produto` ficam na horizontal e a dimensão `Geografia` fica na vertical do relatório. Essa operação é realizada para permitir que os mesmos dados possuam várias visualizações diferentes, para que cada usuário possa escolher aquela que melhor atende às suas necessidades.

Roll-up ou drill-up

A consulta OLAP *roll-up* ou *drill-up* realiza agregações em relação a uma ou várias dimensões. Nessa operação, os dados são sumarizados com generalizações crescentes — por exemplo, vendas diárias são agregadas para vendas mensais, vendas mensais são agregadas para vendas anuais; vendas por UF são agregadas para vendas por região geográfica. A figura 7.24 apresenta um exemplo da consulta *roll-up* ou *drill-up*.

		2021			2022			2023		
		Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira
Sul	PR	3	5	1	2	4	2	3	5	5
	SC	1	4	2	4	9	1	7	6	7
	RS	7	2	6	2	7	4	9	1	6

	2021			2022			2023		
	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira
Sul	11	11	9	8	20	7	19	12	18

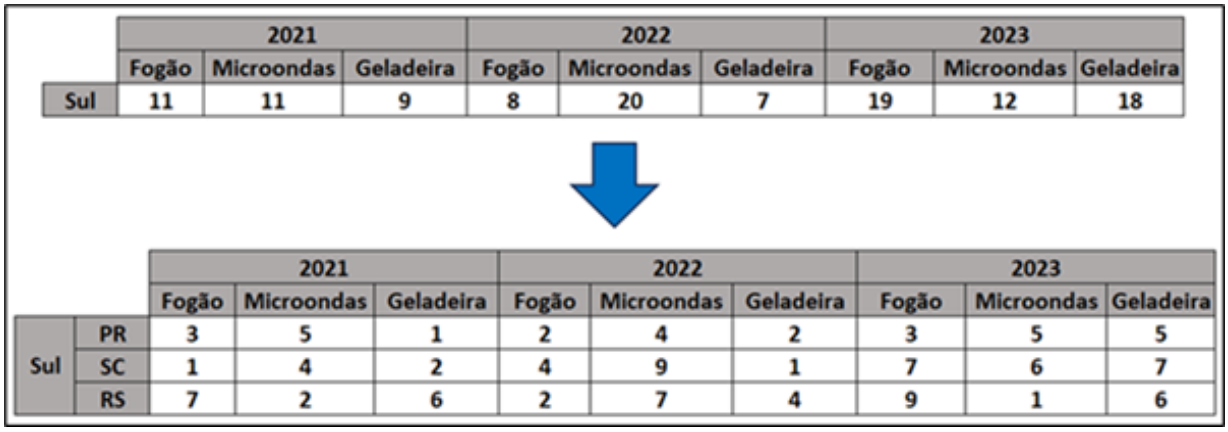
Figura 7.24: Relatório relativo à consulta Roll-up.

Na parte superior da figura 7.24, as dimensões `Tempo` e `Produto` estão na horizontal e a dimensão `Geografia` está na vertical do relatório. Os fatos representam as vendas de cada produto por ano e por UF. Após a execução do *roll-up*, conforme apresentado na parte inferior da figura, as vendas das UFs foram agregadas para vendas por região geográfica — neste caso, a região Sul.

Na parte superior da figura 7.24, na primeira coluna do relatório apresentado, relativa a vendas de fogão no ano de 2021, pode-se observar os valores: 3, para o estado do Paraná; 1, para Santa Catarina; e 7, para o Rio Grande do Sul. Após o *drill-up* ou *roll-up*, pode-se observar no relatório resultante, apresentado na parte inferior da mesma figura, o valor 11 para a venda de fogões na região Sul no ano de 2021, que corresponde à soma das vendas de fogões nos estados do Paraná, Santa Catarina e Rio Grande do Sul.

Drill-down

A consulta OLAP *drill-down* apresenta níveis crescentes de detalhes em relação a uma ou várias dimensões. Nessa operação, os dados são detalhados com desmembramentos. Por exemplo: vendas anuais são desmembradas para vendas mensais; vendas mensais são desmembradas para vendas diárias; vendas por região geográfica são desmembradas para vendas por UF. A figura 7.25 apresenta um exemplo da consulta *drill-down*:



		2021			2022			2023		
		Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira
Sul		11	11	9	8	20	7	19	12	18

		2021			2022			2023		
		Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira	Fogão	Microondas	Geladeira
Sul	PR	3	5	1	2	4	2	3	5	5
	SC	1	4	2	4	9	1	7	6	7
	RS	7	2	6	2	7	4	9	1	6

Figura 7.25: Relatório relativo à consulta Drill-down.

Na parte superior da figura 7.25, as dimensões `Tempo` e `Produto` estão na horizontal e a dimensão `Geografia` está na vertical do relatório. Os fatos representam as vendas de cada produto por ano e por região geográfica. Após a execução do *drill-down*, conforme apresentado na parte inferior da figura, as vendas da região Sul, cujo valor é 11, foram desmembradas para vendas por UFs — neste caso, as UFs da região Sul: Paraná (PR), cujo valor é 3; Santa Catarina (SC), cujo valor é 1; e Rio Grande do Sul (RS), cujo valor é 7.

Ao observar as figuras 7.24 e 7.25, pode-se concluir que a consulta *roll-up* ou *drill-up* segue o caminho inverso da consulta *drill-down*; ou seja, enquanto a operação de *roll-up* ou *drill-up* gera agregações sucessivas, a operação de *drill-down* gera detalhamentos sucessivos.

Embora este capítulo tenha apresentado cada consulta OLAP isoladamente, em aplicações reais, essas consultas são executadas de forma combinada uma ou mais vezes.

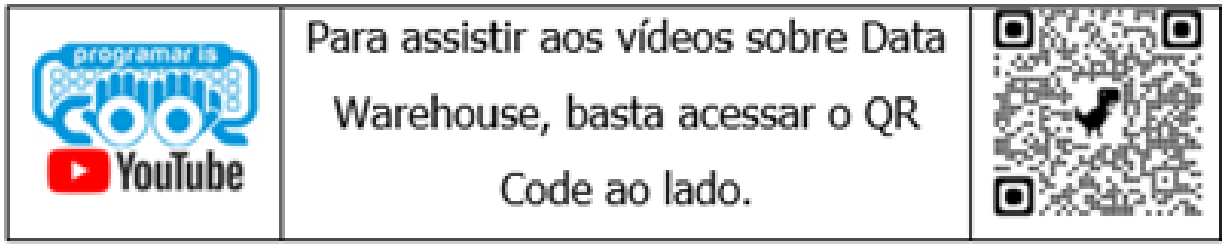


Figura 7.26: Vídeos sobre Data Warehouse. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIDBrGw9wwjIZJLdfacY8BWf

Referências

INMON, W. H. *Como Construir o Data Warehouse*. Rio de Janeiro: Campus, 1997.

KIMBALL, R.; ROSS M.; THORNTHWAITE, W.; MUNDY J.; BECKER, B. *The Date Warehouse Lifecycle Toolkit*. 2. Ed. New York: John Wiley & Sons, 2008.

Capítulo 8
Lógica de Programação

Edsger W. Dijkstra, um cientista da computação holandês, conhecido por suas contribuições nas áreas de desenvolvimento de algoritmos e programas, afirma que *"A arte de programar consiste em organizar e dominar a complexidade"*. Para Aristóteles, renomado filósofo da Grécia Antiga, a Lógica não é ciência e sim um instrumento para o correto pensar. Então, Lógica de Programação pode ser entendida como a técnica de encadear pensamentos para atingir determinado objetivo.

Quando aprendemos Lógica de Programação, na verdade, desenvolvemos a capacidade de dividir um grande problema em um conjunto de pequenos problemas; após esse passo, criamos uma solução para cada pequeno problema; essas soluções, juntas, resolvem o grande problema inicial. Ou seja, consiste em dividir para resolver. Pode-se afirmar que a Lógica de Programação é a base para o aprendizado de todas as linguagens de programação.

Apesar de este livro não possuir o objetivo de formar programadores, decidi abordar Lógica de Programação pelo fato de ser independente de qualquer recurso computacional e devido a sua importância para a alfabetização de dados, pois acredito que todo profissional alfabetizado em dados deve ser capaz de entender um algoritmo, mesmo que superficialmente.

Na verdade, neste capítulo, não apresentarei nada de inédito, mostrarei apenas como expressar de maneira formal o pensamento lógico desenvolvido pelos nossos leitores ao longo de toda a vida. O leitor ou leitora que dominar Lógica de Programação terá a capacidade de aprender qualquer linguagem de programação, pois, embora cada uma possua sua sintaxe peculiar, todas elas são embasadas em Lógica de Programação, ou seja, o primeiro passo

para aprender uma linguagem de programação é entender Lógica de Programação.

Após esta breve introdução, partiremos ao entendimento do que é algoritmo.

8.1 Algoritmo

Algoritmo consiste em um conjunto de regras e operações bem definidas e ordenadas, destinadas à solução de um problema, ou de uma classe de problemas, em um número finito de passos. Algoritmos são criados para automatizar procedimentos repetitivos. De forma genérica, podemos afirmar que as ações de um algoritmo são:

- **Entrada de dados:** o computador recebe dados do mundo externo (usuário).
- **Processamento de dados:** os dados são transformados de acordo com as regras estabelecidas.
- **Saída de dados:** o computador apresenta os dados transformados (resultados) ao mundo externo (usuário).

Existem diversas formas de representação de algoritmos. Não existe um consenso sobre qual é a melhor forma, cada uma tem vantagens e desvantagens. Algumas formas tratam os problemas apenas no nível lógico, abstraindo os detalhes relacionados à implementação; outras formas entram em mais detalhes e são mais específicas. As formas mais conhecidas de representação de algoritmos são:

- Fluxograma;
- Pseudocódigo, também conhecido como “portugol”.

O fluxograma é uma representação gráfica de um procedimento, problema ou sistema, cujas etapas ou módulos são ilustrados de forma encadeada por meio de símbolos geométricos

interconectados. A figura 8.1 apresenta os símbolos utilizados para a construção de fluxogramas. Veremos os mais relevantes com exemplos práticos no decorrer do capítulo.

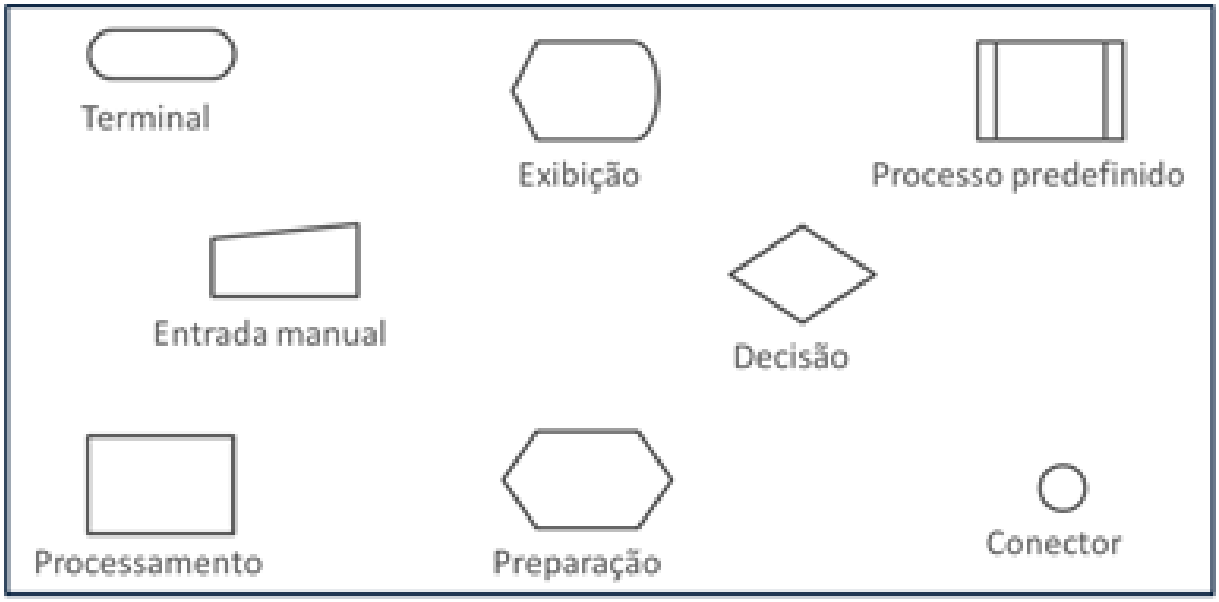


Figura 8.1: Símbolos utilizados em fluxogramas.

A figura 8.2 apresenta um fluxograma para representar um algoritmo que calcula a soma de dois números. Esse algoritmo possui as seguintes ações:

- **Entrada de dados:** o computador recebe dois valores que são atribuídos às variáveis, `a` e `b`.
- **Processamento de dados:** os valores armazenados nas variáveis `a` e `b` são somados e o resultado é atribuído à variável `x`.
- **Saída de dados:** o computador apresenta o valor armazenado na variável `x` ao mundo externo.

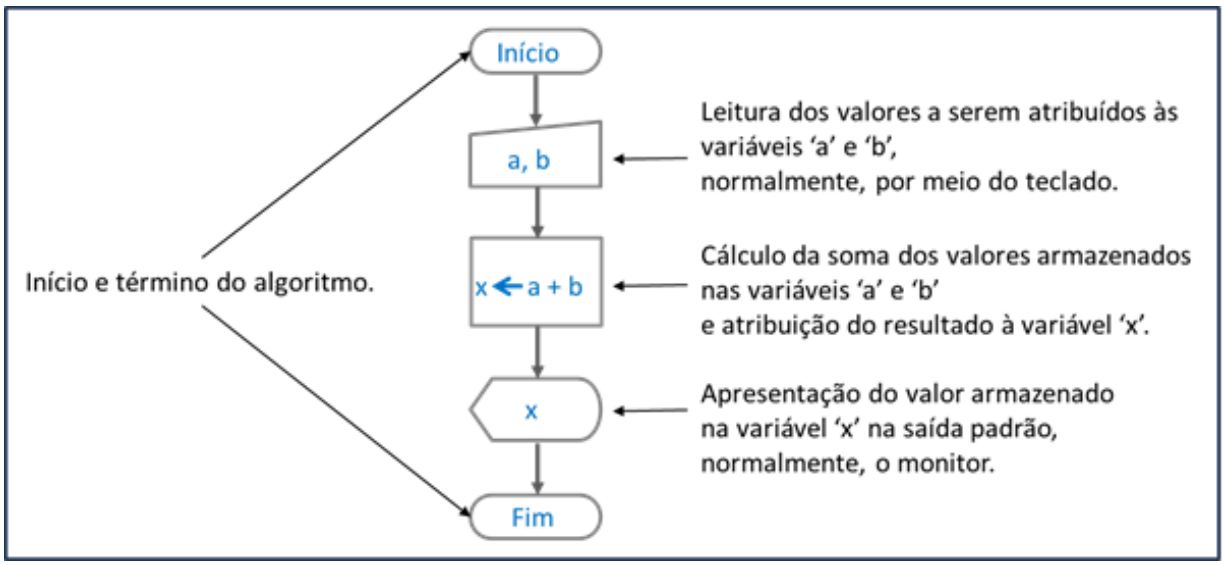


Figura 8.2: Fluxograma para o cálculo da soma de dois números.

Por outro lado, o pseudocódigo é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples, nativa a quem o escreve, de forma a ser entendida por qualquer pessoa, sem a necessidade de conhecer a sintaxe de alguma linguagem de programação. A figura 8.3 apresenta o pseudocódigo para o cálculo da soma de dois números equivalente ao fluxograma da figura 8.2.

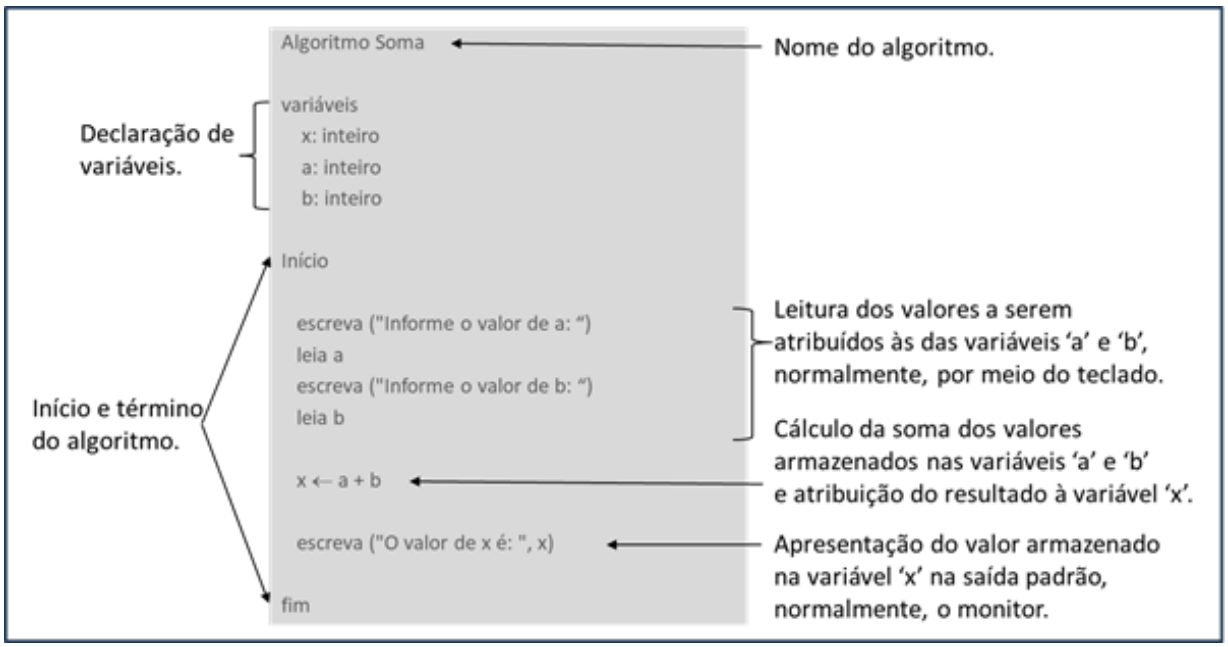


Figura 8.3: Pseudocódigo para o cálculo da soma de dois números.

A figura 8.4 apresenta a interação do usuário com o computador ao executar o algoritmo representado pelo fluxograma da figura 8.2 e pelo pseudocódigo da figura 8.3.

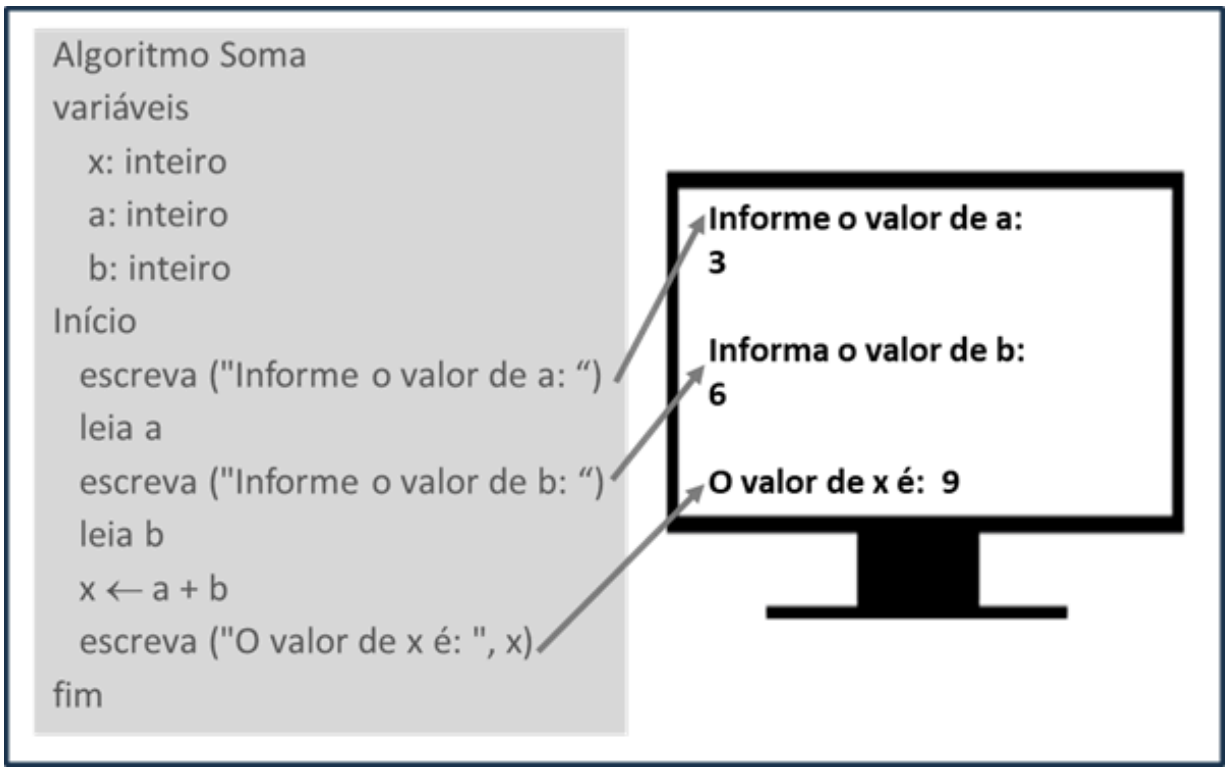


Figura 8.4: Interação do usuário com o computador durante a execução do algoritmo que calcula a soma de dois números.

A figura 8.4 apresenta, à esquerda, o pseudocódigo que calcula a soma de dois números e, à direita, o monitor do computador que exibe a sua execução. O pseudocódigo, inicialmente, declara as variáveis `x`, `a` e `b`, que são do tipo inteiro. Logo após, o comando `escreva (“Informe o valor de a:”)` exibe a mensagem “Informe o valor de a:” no monitor, instruindo o usuário a digitar um número. Após o usuário digitar o número 3 e pressionar a tecla `Enter`, o algoritmo atribui esse valor à variável `a` por meio do comando `leia a`. Após esses passos, o comando `escreva (“Informe o valor de b:”)` exibe a mensagem “Informe o valor de b:” no monitor, instruindo o usuário a digitar um número. Após o usuário digitar o número 6 e pressionar a tecla `Enter`, o algoritmo atribui esse valor à variável `b` por meio do comando `leia b` e executa o próximo passo, que consiste em atribuir à variável `x` o resultado da soma dos valores armazenados nas variáveis `a` e `b`. Após essa atribuição, o algoritmo exibe a

mensagem "O valor de x é:" concatenada ao número armazenado na variável `x` por meio do comando `escreva ("O valor de x é: ", x)`.

Como observado nas figuras 8.2, 8.3 e 8.4, as variáveis desempenhem um papel importante. Uma **variável** é um nome dado a uma região da memória do computador para conter um valor de um determinado tipo. Uma variável deve ser declarada antes de seu uso, o seu valor pode mudar durante o processamento. Por outro lado, uma **constante** é o nome dado a uma região da memória para conter um valor fixo de um determinado tipo. Uma constante também deve ser declarada antes de seu uso.

As variáveis ou constantes podem armazenar os seguintes tipos de dados primitivos ou dados básicos:

- Inteiro: números positivos e negativos não fracionários;
- Real: números inteiros e fracionários positivos e negativos;
- Literal ou caractere: caracteres limitados por aspas ("\"");
- Lógico: verdadeiro ou falso.

As principais regras para atribuirmos o nome a uma variável ou constante são:

- Pode ser constituído por letras do alfabeto (maiúsculas e minúsculas);
- Pode conter dígitos numéricos (0 ... 9);
- Pode conter o caractere underscore (_);
- O primeiro caractere não pode ser um dígito numérico;
- Maiúsculas e minúsculas representam caracteres diferentes, logo, variáveis distintas;
- Uma variável não pode ter por nome uma palavra reservada, ou seja, uma palavra que faça parte da sintaxe de algoritmos, como `se`, `senão`, `enquanto`, `para` etc.

Na construção de um algoritmo, um operador aritmético possui uma precedência sobre os demais, ou seja, uma prioridade para a sua

execução. A prioridade dos operadores aritméticos é a seguinte:

- 1º: Potenciação e radiciação;
- 2º: Multiplicação e divisão;
- 3º: Adição e subtração;
- Operadores com a mesma prioridade são avaliados da esquerda para a direita.

É muito importante entender a precedência dos operadores aritméticos para a construção algoritmos corretos, pois a falta de entendimento dessa precedência pode produzir resultados errados durante a execução de algoritmos. Muitas vezes os parênteses são utilizados em expressões com várias operações aritméticas para que, independentemente da precedência, os operadores aritméticos sejam executados na ordem desejada.

No lado esquerdo da figura 8.5, é apresentado um exemplo de uma expressão composta por várias operações aritméticas executadas de acordo com a precedência dos operadores. Já no lado direito da mesma figura, temos a mesma expressão com operações entre parênteses para que a execução respeite uma determinada regra de negócio.

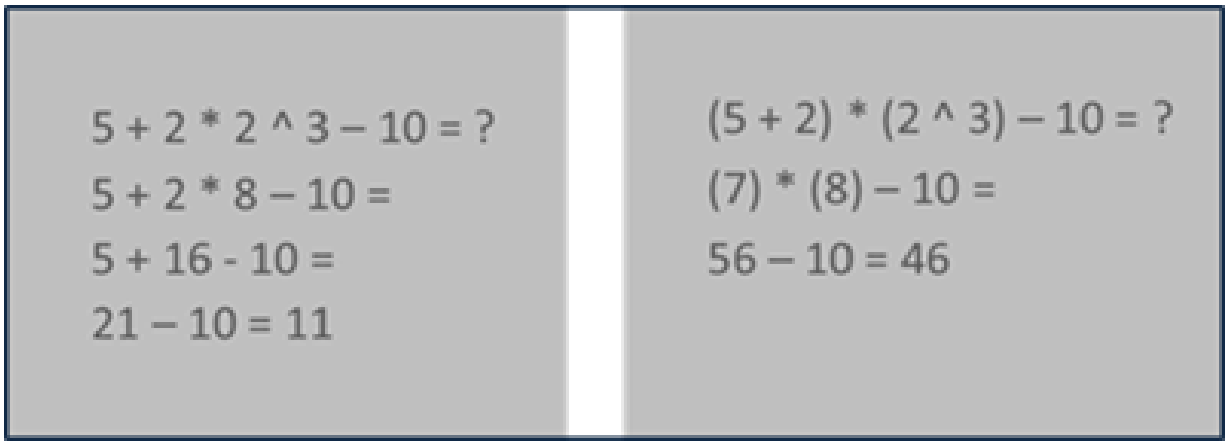


Figura 8.5: Operações aritméticas executadas de formas distintas.

Como pode ser observado na figura 8.5, as operações aritméticas contidas na expressão da esquerda são idênticas à da direita. No

entanto, os resultados da execução das duas expressões são distintos, devido ao fato de a expressão da esquerda ser executada considerando apenas a precedência de operadores, enquanto a da direita utiliza parênteses para respeitar uma determinada regra.

A expressão da esquerda é executada da seguinte forma:

- **Primeira passada:** serão executados os operadores potenciação e radiciação da esquerda para a direita, neste caso apenas a operação 2 ^ 3 (dois elevado ao cubo) = 8.
- **Segunda passada:** serão executados os operadores multiplicação e divisão da esquerda para a direita, neste caso apenas a operação 2 * 8 = 16.
- **Terceira passada:** serão executados os operadores adição e subtração da esquerda para a direita, neste as operações 5 + 16 = 21 e 21 - 10 = 11.

A expressão da direita é executada da seguinte forma:

- **Primeira passada:** serão executadas as operações dentro dos parênteses, neste caso (5 + 2) = 7 e (2 ^ 3) = 8
- **Segunda passada:** por não haver operadores potenciação e radiciação, serão executados os operadores multiplicação e divisão da esquerda para a direita, neste caso apenas a operação 7 * 8 = 56.
- **Terceira passada:** serão executados os operadores adição e subtração da esquerda para a direita, neste a operação 56 - 10 = 46.

Resumidamente, a figura 8.5 deixa claro que podemos organizar as operações aritméticas em um algoritmo da maneira desejada: embora haja uma precedência entre os operadores aritméticos, basta utilizarmos os parênteses para implementarmos a regra desejada.

8.2 Algoritmo com decisão

Em algoritmos, as decisões representam desvios na sua execução. A decisão deve ser baseada em uma ou mais condições. Uma condição é uma expressão lógica (booleana) cujo resultado é um valor falso ou verdadeiro.

Uma expressão logica é obtida por meio de uma relação entre duas variáveis (ou constantes) e um operador relacional. A figura 8.6 apresenta os operadores relacionais.

Operador	Descrição
=	Igual
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
<> ou !=	Diferente de

Figura 8.6: Operadores relacionais.

Quando necessário, podemos usar mais de uma condição em decisões, basta usarmos operadores lógico de conjunção ou disjunção, entre outros, para conectar as condições. A figura 8.7 apresenta a tabela verdade para o operador lógico de conjunção e (.e.).

Condição 1	Condição 2	Resultado (e)
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso
falso	verdadeiro	falso
falso	falso	falso

Figura 8.7: Tabela verdade para o operador lógico de conjunção.

Considerando as seguintes condições e resultado:

- **Condição 1:** seja "dia ensolarado".
- **Condição 2:** seja "ter meio de transporte".
- **Resultado:** seja "ir à praia".

Se utilizarmos essas condições na tabela da figura 8.4 teríamos os seguintes resultados:

- **Linha 1:**
  - **Condição 1:** "dia ensolarado" = verdadeiro;
  - **Condição 2:** "ter meio de transporte" = verdadeiro, então;
  - **Resultado:** "ir à praia" = verdadeiro, ou seja, se for um dia ensolarado e tiver um meio de transporte, irei à praia.
- **Linha 2:**
  - **Condição 1:** "dia ensolarado" = verdadeiro;
  - **Condição 2:** "ter meio de transporte" = falso, então;
  - **Resultado:** "ir à praia" = falso, ou seja, se for um dia ensolarado, mas não tiver um meio de transporte, não irei à praia.
- **Linha 3:**
  - **Condição 1:** "dia ensolarado" = falso;
  - **Condição 2:** "ter meio de transporte" = verdadeiro, então;
  - **Resultado:** "ir à praia" = falso, ou seja, se não for um dia ensolarado mesmo tendo um meio de transporte, não irei à praia.

- **Linha 4:**
  - **Condição 1:** "dia ensolarado" = falso;
  - **Condição 2:** "ter meio de transporte" = falso, então;
  - **Resultado:** "ir à praia" = falso, ou seja, se não for um dia ensolarado e não tiver um meio de transporte, não irei à praia.

Considerando as condições apresentadas anteriormente, pode-se concluir que, para ir à praia, precisa ser um dia ensolarado e ter um meio de transporte. Ou seja, para o resultado ser verdadeiro em uma expressão formada por várias condições utilizando o operador `e`, todas as condições dessa expressão precisam ser verdadeiras; se uma das condições for falsa, o resultado será falso também.

A figura 8.8 apresenta a tabela verdade para o operador lógico de disjunção `ou` (`.ou.`).

Condição 1	Condição 2	Resultado (ou)
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	verdadeiro
falso	verdadeiro	verdadeiro
falso	falso	falso

Figura 8.8: Tabela verdade para o operador lógico de disjunção.

Considerando as seguintes condições e resultado:

- **Condição 1:** seja "ter carona".
- **Condição 2:** seja "ter dinheiro para o táxi".
- **Resultado:** seja "ir ao cinema".

Se Utilizarmos essas condições na tabela da figura 8.8 teríamos os seguintes resultados:

- **Linha 1:**

    - **Condição 1:** "ter carona" = verdadeiro;
    - **Condição 2:** "ter dinheiro para o táxi" = verdadeiro, então;
    - **Resultado:** "ir ao cinema" = verdadeiro, ou seja, se tiver carona e tiver dinheiro para o táxi, irei ao cinema. Repare que neste caso, poderei escolher entre ir ao cinema de carona ou de taxi.
- **Linha 2:**
    - **Condição 1:** "ter carona" = verdadeiro;
    - **Condição 2:** "ter dinheiro para o táxi" = falso, então;
    - **Resultado:** "ir ao cinema" = verdadeiro, ou seja, se tiver carona, mas não tiver dinheiro para o táxi, irei ao cinema. Repare que neste caso irei ao cinema de carona.
- **Linha 3:**
    - **Condição 1:** "ter carona" = falso;
    - **Condição 2:** "ter dinheiro para o táxi" = verdadeiro, então;
    - **Resultado:** "ir ao cinema" = verdadeiro, ou seja, se não tiver carona, mas tiver dinheiro para o táxi, irei ao cinema. Repare que neste caso irei ao cinema de taxi.
- **Linha 4:**
    - **Condição 1:** "ter carona" = falso;
    - **Condição 2:** "ter dinheiro para o táxi" = falso, então;
    - **Resultado:** "ir ao cinema" = falso, ou seja, se não tiver carona nem dinheiro para o táxi, não irei ao cinema.

Assim, considerando as condições apresentadas anteriormente, pode-se concluir que, para ir ao cinema, basta satisfazer apenas uma das condições, "ter carona" ou "ter dinheiro para o táxi". Ou seja, para o resultado ser verdadeiro em uma expressão formada por várias condições utilizando o operador `ou` , basta apenas uma das condições ser verdadeira.

Após entendermos o uso de operadores lógicos de conjunção e disjunção em expressões lógicas apresentados nesta seção, nas

próximas seções vamos entender como eles são utilizados nos desvios condicionais contidos em algoritmos.

Desvio condicional simples

O desvio condicional simples é utilizado quando uma estrutura de decisão possui comandos a serem executados somente quando a condição testada for verdadeira, caso contrário, a execução do algoritmo vai para o fim da estrutura de decisão sem executar determinados comandos.

A figura 8.9 apresenta um exemplo de fluxograma de um desvio condicional simples com o pseudocódigo equivalente ao lado.

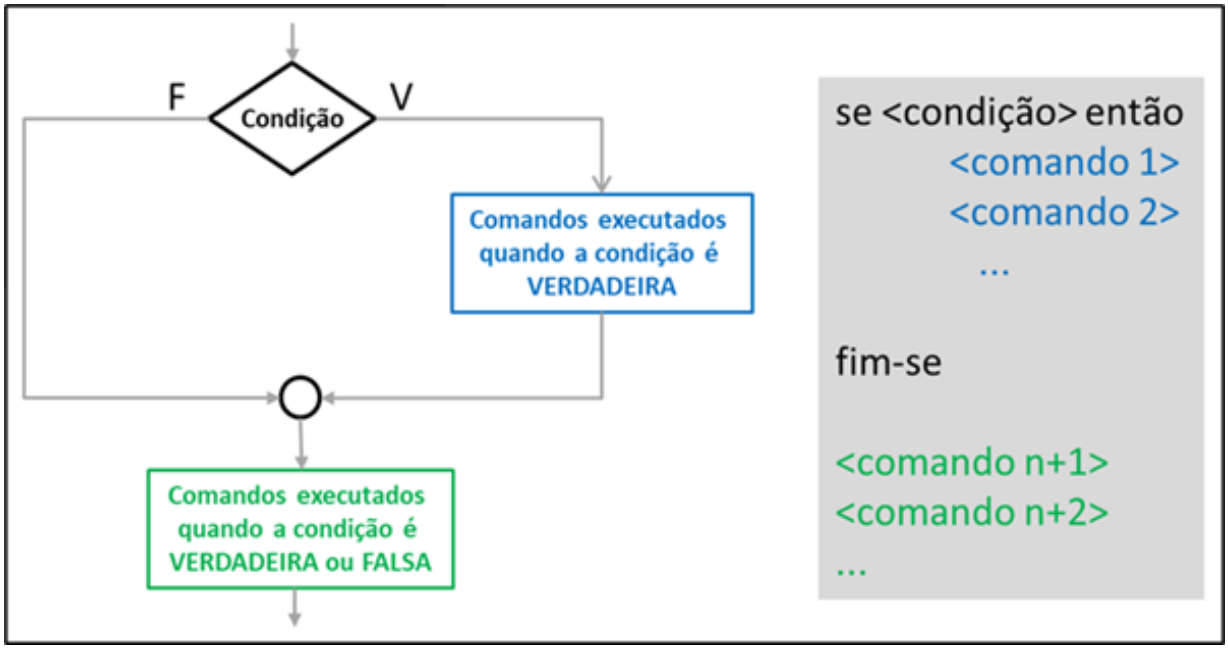


Figura 8.9: Desvio condicional simples.

Como pode ser observado no fluxograma apresentado na figura 8.9, após testar a condição, que pode ser verdadeira ou falsa, a execução do algoritmo seguirá fluxos diferentes de acordo com o resultado desse teste.

Caso a condição seja verdadeira, o algoritmo executará os comandos existentes no retângulo à direita e depois irá para o fim da estrutura

de decisão representada pelo círculo. Após esse passo, o algoritmo executará os comandos existentes no retângulo abaixo do círculo.

Por outro lado, caso a condição seja falsa, o algoritmo irá diretamente para o fim da estrutura de decisão representada pelo círculo. Após esse passo, também executará os comandos existentes no retângulo abaixo do círculo, pois esses comandos estão fora da estrutura de decisão.

À direita da figura 8.9 é apresentado o pseudocódigo equivalente ao fluxograma. Nele, a condição a ser testada está no início do desvio condicional simples representada pelos termos `se <condição> então` e o fim desse desvio condicional está representado pelo termo `fim-se`.

Quando a execução do algoritmo atinge a linha `se <condição> então`, se a `condição` for verdadeira, então, o algoritmo executará os comandos imediatamente abaixo dessa instrução e depois irá para o fim da estrutura de decisão, ou seja, a linha `fim-se`. Após esse passo, o algoritmo executará os comandos imediatamente abaixo da instrução `fim-se`.

Por outro lado, se a `condição` for falsa, o algoritmo irá diretamente para a linha `fim-se`, pulando os comandos imediatamente abaixo da instrução `se <condição> então`. Após esse passo, também executará os comandos imediatamente abaixo da instrução `fim-se`, pois esses comandos estão fora da estrutura de decisão.

Exemplo de algoritmo com desvio condicional simples

Enunciado: implemente um algoritmo com um desvio condicional simples que receba dois números e teste se a soma desses dois números é maior que 10. Caso o resultado desse teste seja verdadeiro, o algoritmo deve exibir a soma, caso contrário, nenhum número será exibido. Quando finalizar o desvio condicional, o algoritmo deverá imprimir a frase "Estou fora da estrutura de decisão" antes de finalizar sua execução.

A figura 8.10 apresenta o fluxograma e o pseudocódigo que implementam a solução do enunciado anterior.

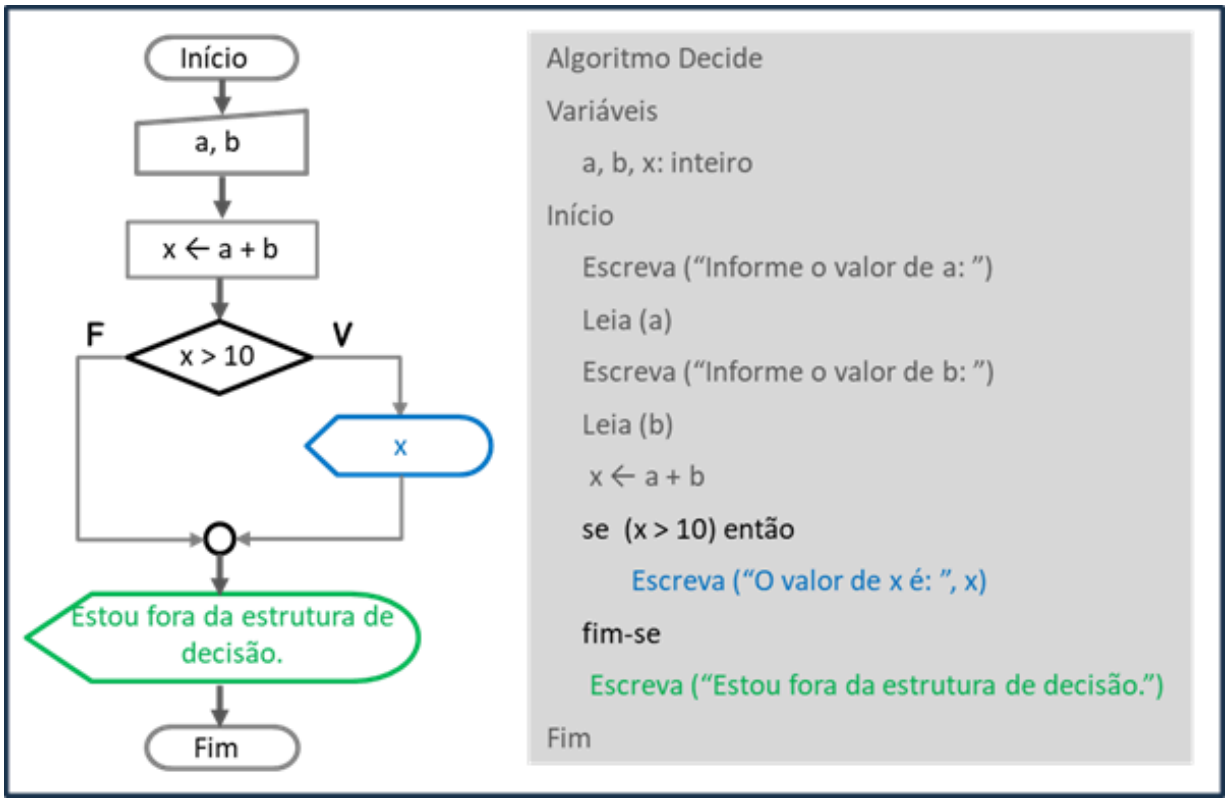


Figura 8.10: Exemplo de algoritmo com desvio condicional simples.

Como pode ser observado no fluxograma e no pseudocódigo da figura 8.10, inicialmente são declaradas as variáveis `a`, `b` e `x` (pseudocódigo), todas do tipo inteiro. Após a declaração das variáveis, inicia-se a execução do algoritmo atribuindo valores às variáveis `a` e `b` (o trapézio representa entrada de dados). Em seguida, a variável `x` recebe a soma das variáveis `a` e `b` (o retângulo representa o processamento das operações sobre os dados).

Depois desse passo, é realizado o teste para saber se `x` é maior que 10 (o losango representa o início da estrutura de decisão com a condição a ser testada). Caso o resultado do teste seja verdadeiro, o algoritmo seguirá o fluxo da direita e exibirá `x` no monitor do computador onde o algoritmo está sendo executado (a figura

geométrica representa exibição) e depois finalizará a estrutura de decisão (representada pelo círculo). Por outro lado, caso o resultado do teste seja falso, o algoritmo caminhará diretamente para o fim da estrutura de decisão (representada pelo círculo).

Após o fim da estrutura de decisão (representada pelo círculo), o algoritmo exibirá a frase "Estou fora da estrutura de decisão" no monitor (a figura geométrica representa exibição), ou seja, independentemente de o resultado do teste ser verdadeiro ou falso o algoritmo sempre apresentará a frase "Estou fora da estrutura de decisão", pois esse comando está após o fim da estrutura de decisão. Por fim, a execução do algoritmo é finalizada.

Desvio condicional composto

O desvio condicional composto é utilizado quando uma estrutura de decisão possui comandos distintos a serem executados dependendo se a condição testada for verdadeira ou falsa.

A figura 8.11 apresenta um exemplo de fluxograma de um desvio condicional composto com o pseudocódigo equivalente ao lado.

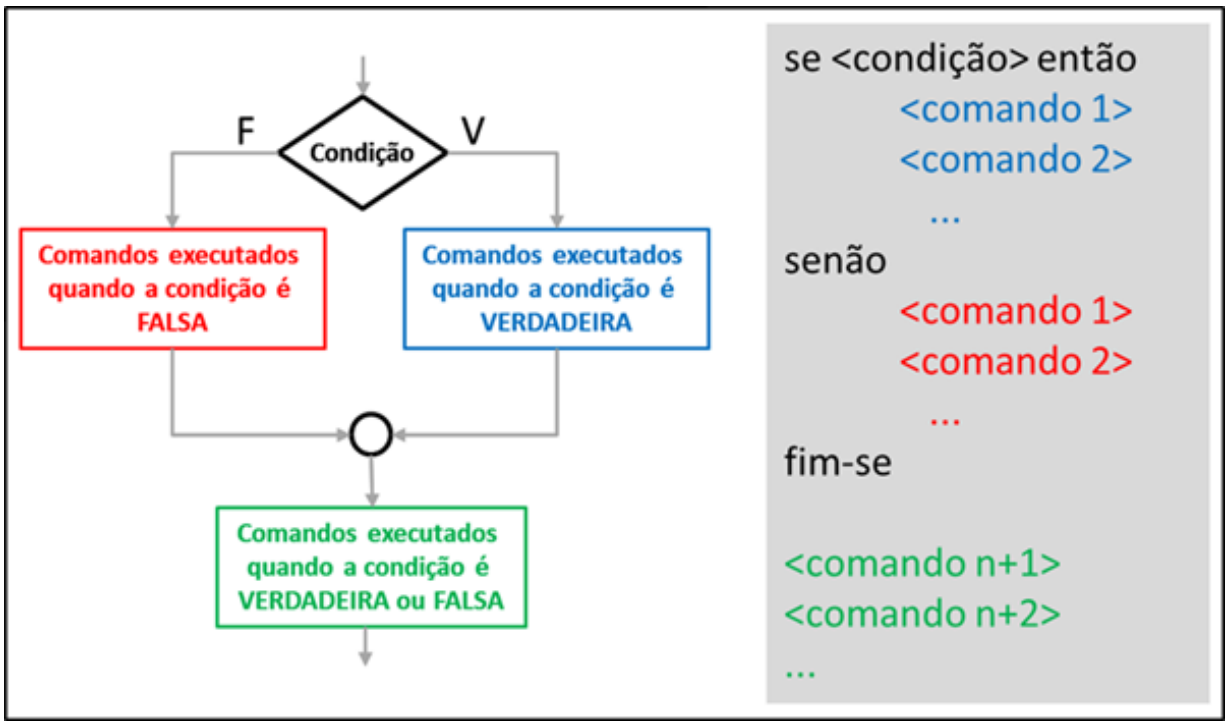


Figura 8.11: Desvio condicional composto.

Como pode ser observado no fluxograma apresentado na figura 8.11, após testar a condição, que pode ser verdadeira ou falsa, a execução do algoritmo seguirá fluxos diferentes de acordo com o resultado desse teste.

Caso a condição seja verdadeira, o algoritmo executará os comandos existentes no retângulo à direita e depois irá para o fim da estrutura de decisão representada pelo círculo. Após esse passo, o algoritmo executará os comandos existentes no retângulo após o círculo.

Por outro lado, caso a condição seja falsa, o algoritmo executará os comandos existentes no retângulo à esquerda e depois irá para o fim da estrutura de decisão representada pelo círculo. Após esse passo, também executará os comandos existentes no retângulo após o círculo, pois esses comandos estão fora da estrutura de decisão.

À direita da figura 8.11, é apresentado o pseudocódigo equivalente ao fluxograma. Nele, a condição a ser testada está no início do desvio condicional composto, representado pelos termos `se`

`<condição> então` , pelo desvio representado pelo termo `senão` e o fim desse desvio condicional está representado pelo termo `fim-se` .

Quando a execução do algoritmo atinge a linha `se <condição> então` , se a condição for verdadeira, então, o algoritmo executará os comandos imediatamente após essa estrutura até a linha `<senão>` e depois irá para o fim da estrutura de decisão, ou seja, a linha `fim-se` e executará os comandos imediatamente após.

Por outro lado, se a condição for falsa, o algoritmo executará os comandos após a linha `senão` e depois irá para o fim da estrutura de decisão, ou seja, a linha `fim-se` . Após esse passo, também executará os comandos após a linha `fim-se` , pois esses comandos estão fora da estrutura de decisão.

Exemplo de algoritmo com desvio condicional composto

Enunciado: implemente um algoritmo com um desvio condicional composto que receba dois números e teste se a soma desses dois números é maior que 10. Caso o resultado desse teste seja verdadeiro, o algoritmo deve exibir o valor do primeiro número digitado, caso contrário, o algoritmo deve exibir o valor do segundo número digitado. Quando finalizar o desvio condicional, o algoritmo deverá imprimir a frase "Estou fora da estrutura de decisão" antes de finalizar sua execução.

A figura 8.12 apresenta o fluxograma e o pseudocódigo que implementam a solução do enunciado anterior.

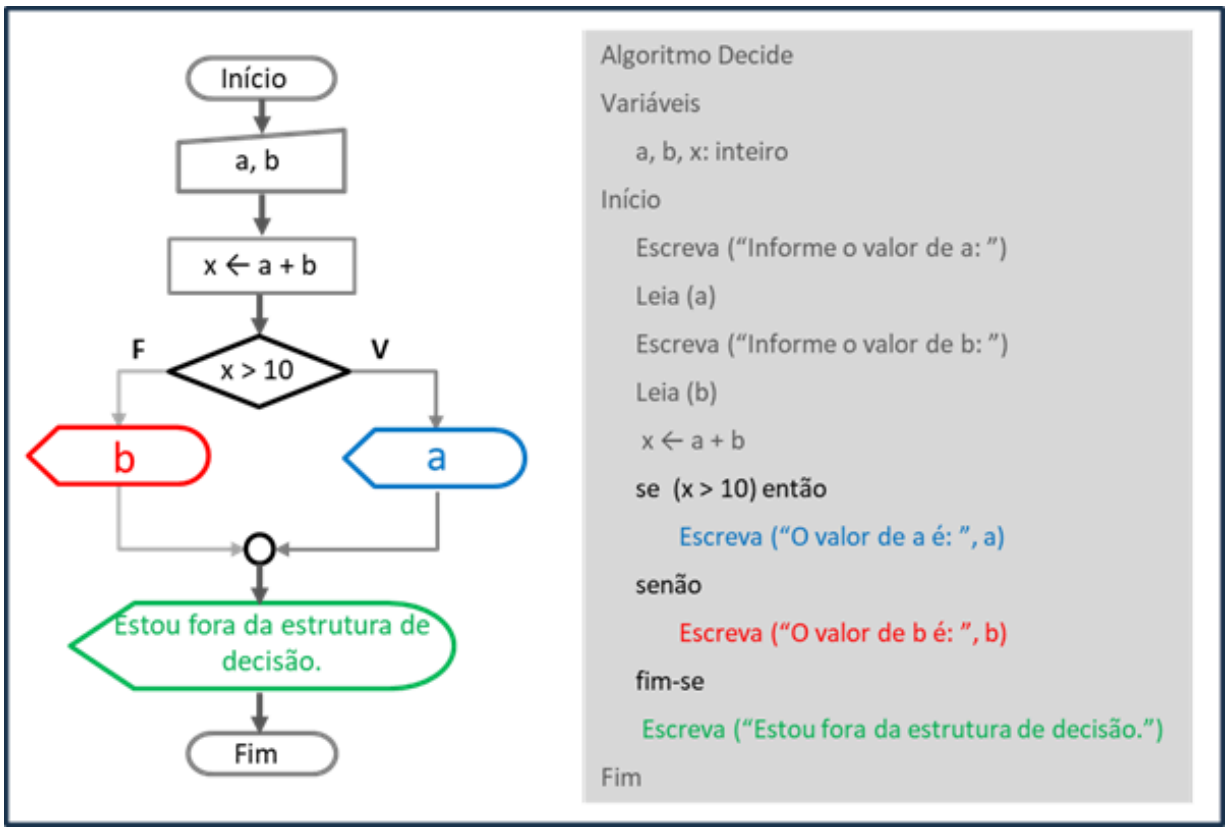


Figura 8.12: Exemplo de algoritmo com desvio condicional composto.

Como pode ser observado no fluxograma e no pseudocódigo da figura 8.12, inicialmente são declaradas as variáveis `a`, `b` e `x` (pseudocódigo), todas do tipo inteiro. Após a declaração das variáveis, inicia-se a execução do algoritmo atribuindo valores às variáveis `a` e `b` (o trapézio representa entrada de dados). Em seguida, a variável `x` recebe a soma das variáveis `a` e `b` (o retângulo representa o processamento das operações sobre os dados).

Depois desse passo, é realizado o teste para saber se `x` é maior que 10 (o losango representa o início da estrutura de decisão com a condição a ser testada). Caso o resultado do teste seja verdadeiro, o algoritmo seguirá o fluxo da direita e exibirá o valor armazenado na variável `a` no monitor do computador onde o algoritmo está sendo executado (a figura geométrica representa exibição) e depois finalizará a estrutura de decisão (representada pelo círculo). Por

outro lado, caso o resultado do teste seja falso, o algoritmo seguirá o fluxo da esquerda e exibirá o valor armazenado na variável `b` no monitor do computador onde o algoritmo está sendo executado (a figura geométrica representa exibição) e depois finalizará a estrutura de decisão (representada pelo círculo).

Após o fim da estrutura de decisão (representada pelo círculo), o algoritmo exibirá a frase "Estou fora da estrutura de decisão" no monitor (a figura geométrica representa exibição), ou seja, independentemente de o resultado do teste ser verdadeiro ou falso o algoritmo sempre apresentará a frase "Estou fora da estrutura de decisão", pois esse comando está após o fim da estrutura de decisão. Por fim, a execução do algoritmo é finalizada.

Depois de entendermos os desvios condicionais contidos em algoritmos apresentados nesta seção, nas próximas seções vamos entender como as estruturas de repetições são implementadas em algoritmos.

8.3 Algoritmo com repetição

O recurso de repetição em algoritmos consiste em uma forma de executar blocos de comandos somente sob determinadas condições, mas com a opção de repetir o mesmo bloco quantas vezes for necessário. Cada vez que o conjunto de ações (comandos) é executado, acontece uma iteração. As estruturas de repetição também são chamadas de laços, loopings ou loops.

Essas estruturas de programação proporcionam a repetição de determinados trechos de código. É uma técnica utilizada para reduzir o trabalho de programação, principalmente quando é necessário repetir várias vezes alguma ação do programa.

Os principais laços são:

- Incondicional ou contado;
- Condicional com teste no início;
- Condicional com teste no fim.

Laço incondicional ou contado

O laço incondicional ou contado é utilizado quando é possível saber, de antemão, quantas vezes será necessário repetir um conjunto de comandos em um algoritmo. Esse número de repetições pode ser um valor fixo, pode ser um valor informado pelo usuário, ou calculado pelo programa, porém o valor sempre é conhecido antes de iniciar o laço. A figura 8.13 apresenta um exemplo de fluxograma de um laço incondicional ou contado com o pseudocódigo equivalente ao lado.

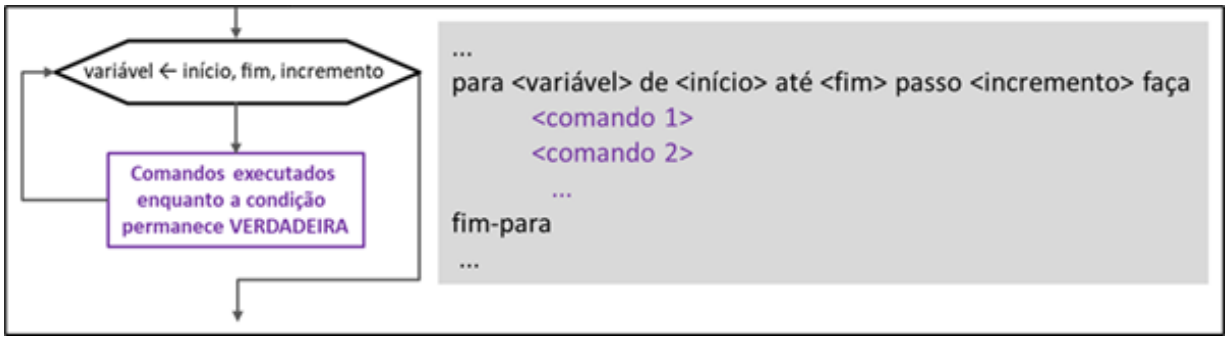


Figura 8.13: Laço incondicional ou contado.

Como pode ser observado no fluxograma apresentado na figura 8.13, a estrutura de laço incondicional ou contado é composta por um teste no seu início, representado pelo hexágono, e um conjunto de comandos, representado pelo retângulo, que serão executados várias vezes, cada execução desse conjunto de comandos caracteriza uma iteração. O término da execução acontece quando uma variável, que é incrementada a cada iteração, assume um determinado valor.

O teste do laço é formado por uma variável que recebe um valor inicial (início), essa variável é incrementada a cada iteração (execução dos comandos internos ao laço) por um valor estabelecido

(incremento) até ultrapassar o valor limite (fim), quando esse valor é ultrapassado, a execução do algoritmo sai do laço para dar continuidade ao algoritmo.

Então, o teste consiste em verificar se a variável, que recebeu um valor inicial, ultrapassou o valor limite a cada iteração, tendo em vista que a cada iteração essa variável é incrementada por um valor preestabelecido, o incremento.

À direita da figura 8.13, é apresentado o pseudocódigo equivalente ao fluxograma. Nele, a escolha da variável (`<variável>`), a atribuição do valor inicial (`<início>`), o valor limite (`<fim>`) e o incremento (`<incremento` >) são representados pelos termos `para <variável> de <início> até <fim> passo <incremento> faça` . O conjunto de comandos que pertencem ao laço é limitado pelos termos `fim-para` .

Quando a execução do algoritmo atinge a linha `para <variável> de <início> até <fim> passo <incremento> faça` pela primeira vez, a variável escolhida recebe o valor inicial e executa os comandos internos ao laço.

Após essa execução, a variável escolhida é incrementada pelo valor do incremento e o algoritmo testa se o valor da variável ultrapassou o valor limite e, caso não tenha ultrapassado, os comandos internos ao laço são executados novamente e a variável é incrementada. Esses comandos são limitados pela linha `fim-para` .

Por outro lado, caso o valor da variável tenha ultrapassado o valor limite, o algoritmo sai do laço e executa os comandos externos ao laço, ou seja, após a linha `fim-para` , e finaliza a execução.

Exemplo de algoritmo com laço incondicional ou contado

Enunciado: implemente um algoritmo utilizando laço incondicional ou contado que apresente no monitor do usuário os valores inteiros de 1 até 3.

A figura 8.14 apresenta o fluxograma e o pseudocódigo que implementam a solução do enunciado anterior.

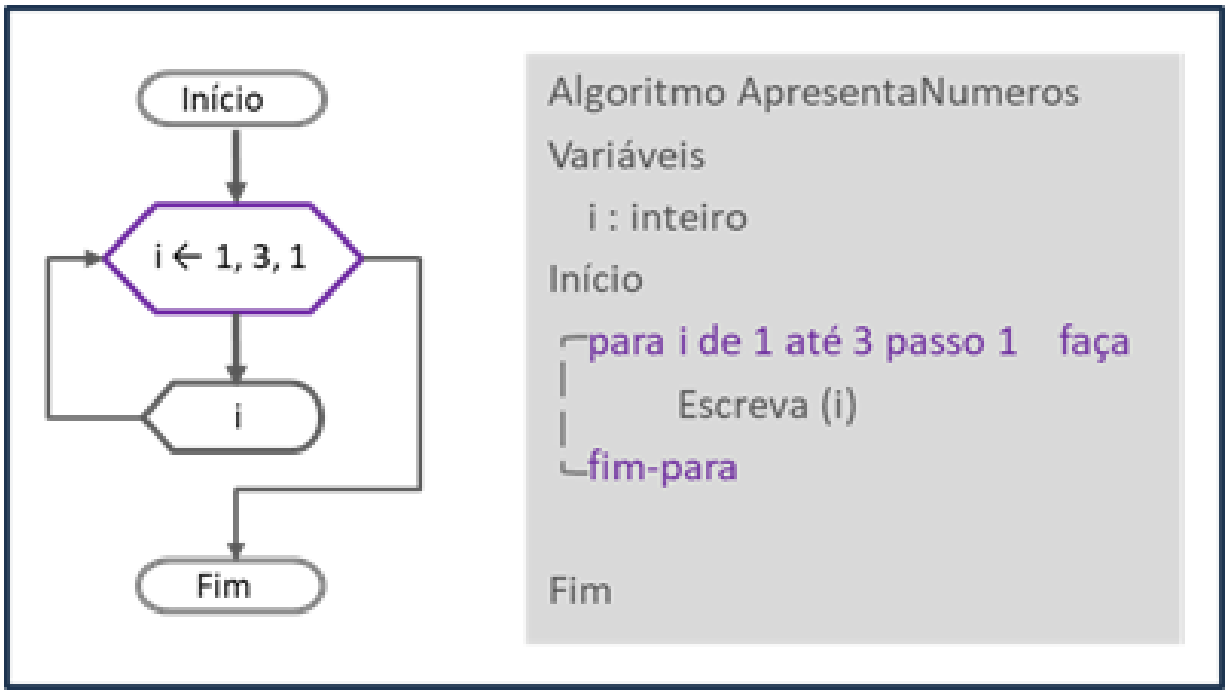


Figura 8.14: Exemplo de algoritmo com laço incondicional ou contado.

Como pode ser observado no fluxograma e no pseudocódigo da figura 8.14, inicialmente é declarada a variável `i` (pseudocódigo), do tipo inteiro. Após a declaração dessa variável, inicia-se a execução do algoritmo atribuindo o número 1 à variável `i`, estabelecendo o número 3 como limite e o número 1 como incremento (o hexágono contém essas informações). No pseudocódigo, essas informações estão na linha `para i de 1 até 3 passo 1 faça`.

Após a variável `i` receber o valor 1, o algoritmo executará os comandos internos ao laço, limitados pela linha `fim-para`, neste caso existe apenas um comando que consiste na exibição do valor armazenado na variável `i` no monitor do computador onde o algoritmo está sendo executado (a figura geométrica representa exibição). Após esse passo, a variável `i` é incrementada pelo valor 1 e volta para o teste do laço (representada pelo hexágono) para

verificar se a variável `i` superou o valor 3, caso o valor seja inferior ou igual a 3, o algoritmo executará o comando de exibição da variável `i` novamente, caso o valor seja superior, o laço será encerrado e o algoritmo será finalizado. Assim, serão exibidos no monitor os valores 1, 2, 3, conforme a solicitação do enunciado.

A seguir, será apresentada a execução do algoritmo implementado. Cabe ressaltar que serão apresentados apenas os números de 1 a 3 para facilitar a explicação da execução do algoritmo, porém poderíamos utilizar o mesmo algoritmo para apresentar qualquer intervalo de números, basta mudarmos os valores iniciais e finais da variável `i` na linha `para <variável> de <início> até <fim> passo <incremento> faça`. A figura 8.15 apresenta a declaração da variável `i` pelo algoritmo.

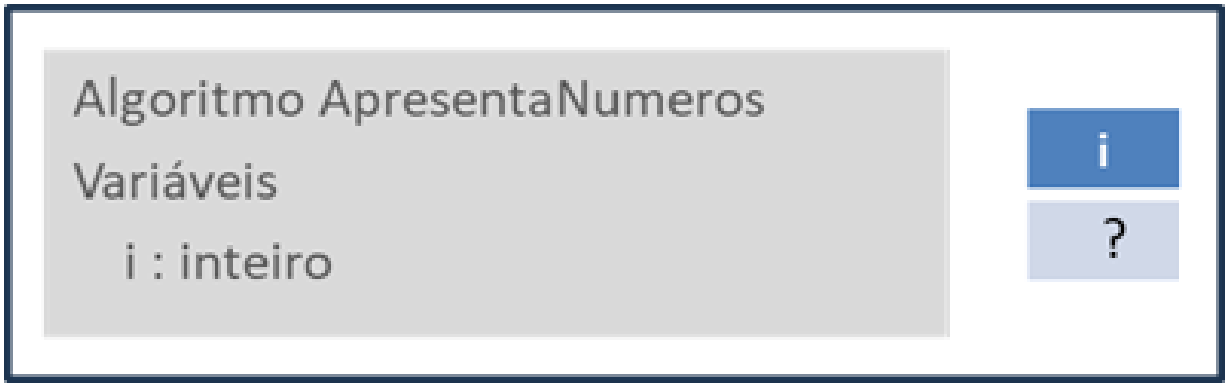


Figura 8.15: Declaração da variável i.

O lado esquerdo da figura 8.15 apresenta a parte do pseudocódigo responsável pela declaração da variável `i`. A tabela do lado direto dessa figura representa a memória do computador, pois uma variável é uma referência a uma área de memória do computador. O conteúdo armazenado pela variável `i` é uma interrogação, que representa a incerteza do conteúdo da área de memória referenciada pela variável, pois essa área de memória pode ter sido utilizada por outro programa e pode ter ficado com algum valor que é desconhecido para o pseudocódigo atual. A figura 8.16 apresenta a primeira execução do laço.

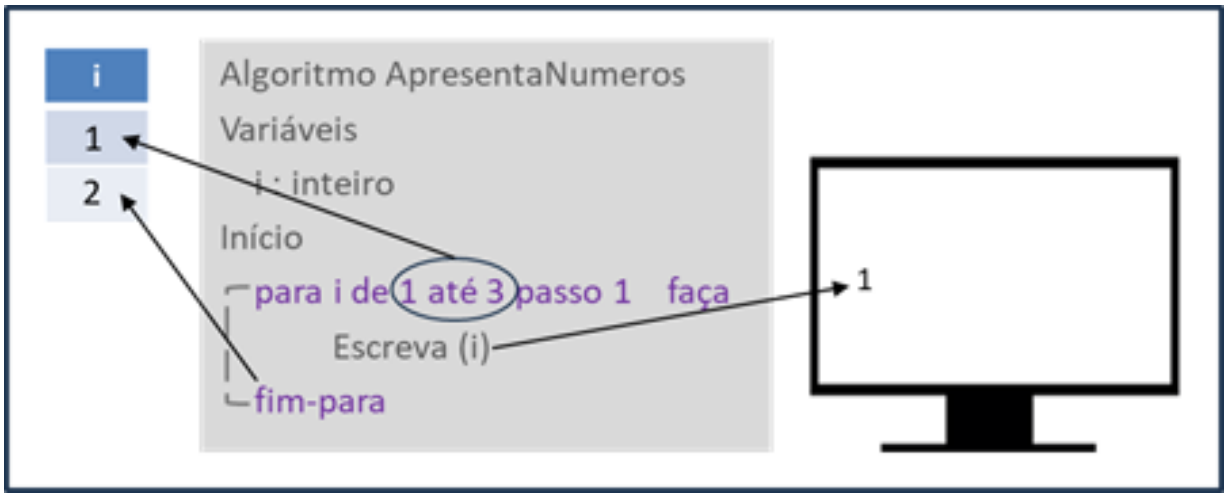


Figura 8.16: Primeira execução do laço incondicional.

Segundo a figura 8.16, na primeira execução do laço incondicional, a variável `i` receberá o número 1, conforme a tabela que representa a memória do computador. Após esse passo, o algoritmo testará se o valor armazenado em `i`, o valor 1, é menor ou igual a 3, pois os comados internos ao laço serão executados até o valor de `i` atingir o valor 3. Como 1 é menor que 3, o comando interno ao laço será executado.

Esse comando consiste em apresentar o conteúdo armazenado na variável `i` no monitor do computador onde o algoritmo está sendo executado. Assim, aparecerá no monitor o número 1 e, antes de atingir o fim do laço o algoritmo incrementará o valor da variável `i` de uma unidade, que é o valor do `passo` (incremento), passando a armazenar o valor 2. Após esse passo a execução retornará ao início do laço para uma nova execução. A figura 8.17 apresenta a segunda execução do laço.

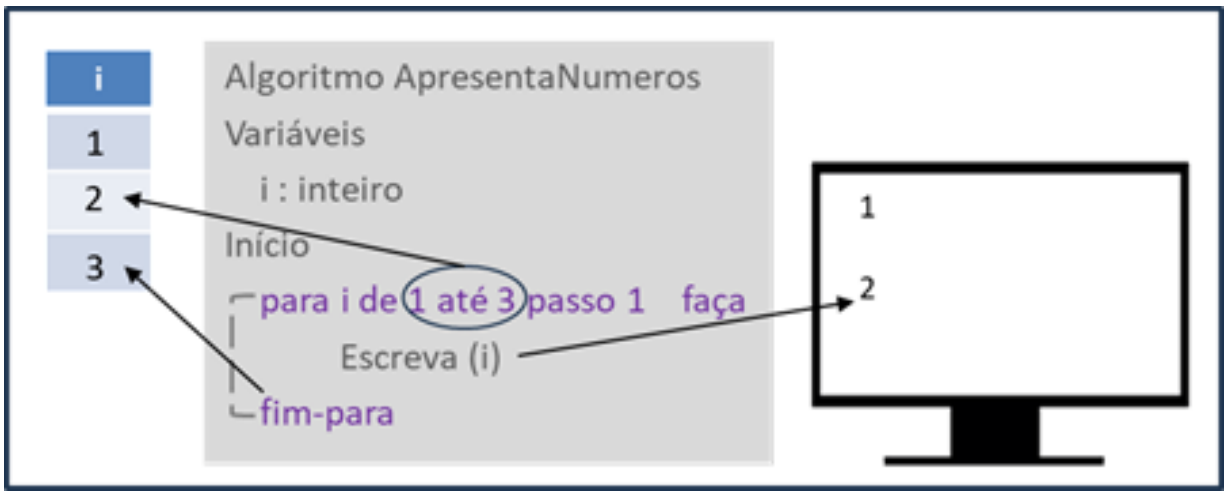


Figura 8.17: Segunda execução do laço incondicional.

Segundo a figura 8.17, na segunda execução do laço incondicional, o algoritmo testará se o valor armazenado em `i`, o valor 2, é menor ou igual a 3, pois os comados internos ao laço serão executados até o valor de `i` atingir o valor 3. Como 2 é menor que 3, o comando interno ao laço será executado.

Esse comando consiste em apresentar o conteúdo armazenado na variável `i` no monitor do computador onde o algoritmo está sendo executado. Assim, aparecerá no monitor o número 2 e, antes de atingir o fim do laço o algoritmo incrementará o valor da variável `i` de uma unidade, que é o valor do `passo` (incremento), passando a armazenar o valor 3. Após esse passo a execução retornará ao início do laço para uma nova execução. A figura 8.18 apresenta a terceira execução do laço.

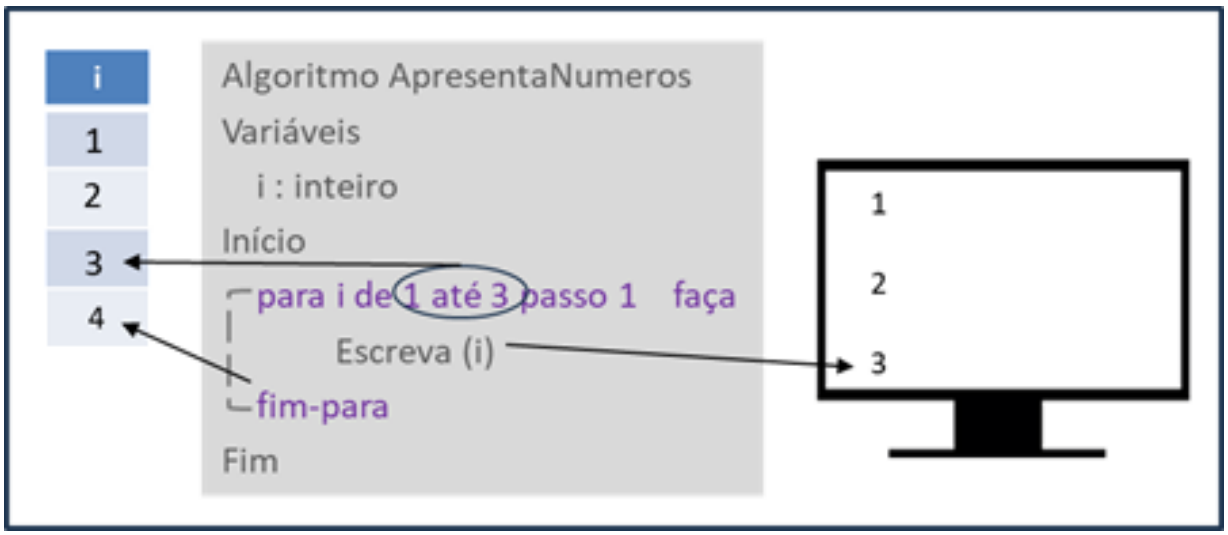


Figura 8.18: Terceira execução do laço incondicional.

Segundo a figura 8.18, na terceira execução do laço incondicional, o algoritmo testará se o valor armazenado em `i`, que é igual a 3, pois os comados internos ao laço serão executados até o valor de `i` atingir o valor 3. Assim, o comando interno ao laço será executado.

Esse comando consiste em apresentar o conteúdo armazenado na variável `i` no monitor do computador onde o algoritmo está sendo executado. Assim, aparecerá no monitor o número 3 e, antes de atingir o fim do laço o algoritmo incrementará o valor da variável `i` de uma unidade, que é o valor do `passo` (incremento), passando a armazenar o valor 4. Após esse passo a execução retornará ao início do laço para uma nova execução.

No entanto, como a variável `i` armazena o valor 4, o laço não será mais executado, pois o limite máximo do valor de `i` para permanecer executando o laço é 3. Assim, o algoritmo sairá do laço e finalizará a sua execução.

Laço condicional com teste no início

O laço condicional com teste no início começa a execução de um bloco de comandos quando uma condição é verdadeira. Neste caso,

não é preciso saber, de antemão, quantas vezes será necessário repetir um conjunto de comandos em um algoritmo.

Esse laço repete a execução desse bloco enquanto a condição for verdadeira, quando a condição se torna falsa, a execução do bloco é encerrada. A figura 8.19 apresenta um exemplo de fluxograma de um laço condicional com teste no início com o pseudocódigo equivalente ao lado.

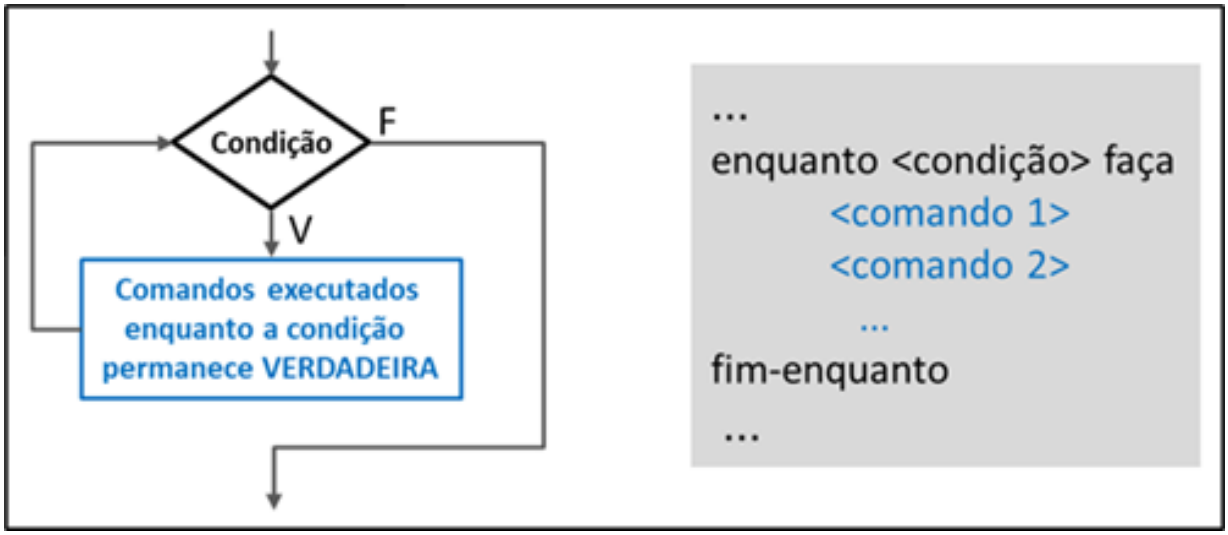


Figura 8.19: Laço condicional com teste no início.

Como pode ser observado no fluxograma apresentado na figura 8.19, a estrutura de laço condicional com teste no início é composta pelo teste de uma condição no seu início e um conjunto de comandos que serão executados várias vezes. Cada execução desse conjunto de comandos caracteriza uma iteração. O término da execução acontece quando a condição testada se torna falsa.

O início do laço possui uma condição que pode assumir o valor verdadeiro ou falso, enquanto essa condição for verdadeira, o conjunto de comandos que estão no interior do laço serão executados; porém, quando essa condição se tornar falsa, a execução do algoritmo sai do laço para dar continuidade ao algoritmo. Então, o teste consiste em verificar se a condição do início do laço é verdadeira ou falsa.

À direita da figura 8.19, é apresentado o pseudocódigo equivalente ao fluxograma. Nele, a condição a ser testada compõe o termo `enquanto <condição> faça`.

Quando a execução do algoritmo atinge a linha `enquanto <condição> faça`, a condição é testada, caso o resultado do teste seja verdadeiro, o algoritmo executa os comandos internos ao laço que são limitados pela linha `fim-enquanto`.

Após cada execução do bloco de comandos, o algoritmo testa se a condição é verdadeira, caso positivo, os comandos internos ao laço são executados novamente. Por outro lado, caso a condição testada seja falsa, o algoritmo sai do laço e executa os comandos externos e finaliza a execução do algoritmo.

Exemplo de algoritmo com laço condicional com teste no início

Enunciado: implemente um algoritmo utilizando laço condicional com teste no início que leia a idade de várias pessoas e no final apresente a média das idades. O critério de parada será a digitação de uma idade igual a zero ou negativa, ou seja, uma idade menor ou igual a zero.

A figura 8.20 apresenta o fluxograma e o pseudocódigo que implementam a solução do enunciado anterior utilizando laço condicional com teste no início.

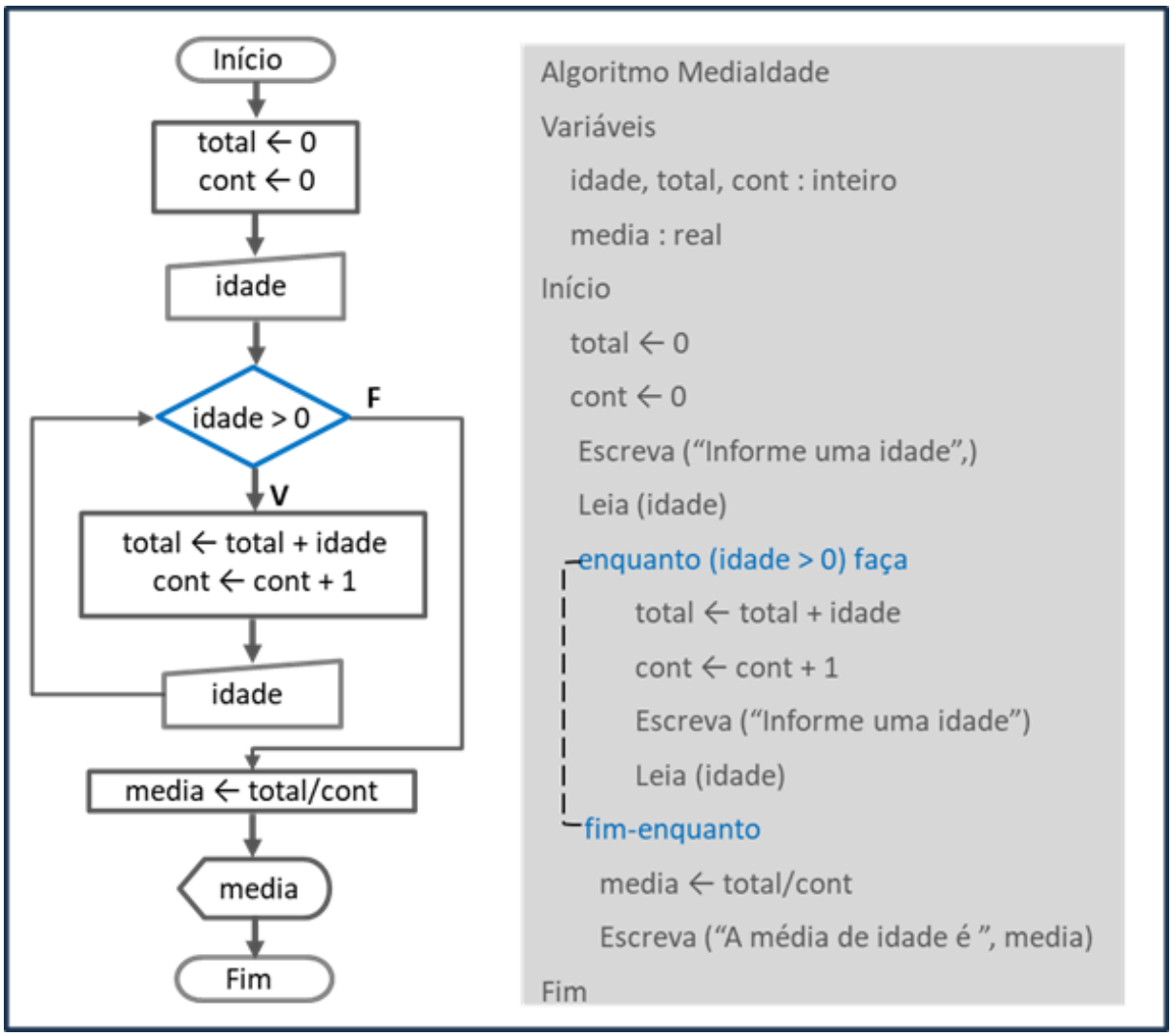


Figura 8.20: Exemplo de algoritmo com laço condicional com teste no início.

Como pode ser observado no fluxograma e no pseudocódigo da figura 8.20, inicialmente são declaradas as variáveis `idade`, `total`, `cont`, todas do tipo inteiro, e a variável `media`, do tipo real. A variável `idade` armazenará cada idade digitada pelo usuário, a variável `total` armazenará a soma de todas as idades digitadas, a variável `cont` armazenará a quantidade de idades digitadas e a variável `média` armazenará a média das idades digitadas. Esse valor será obtido dividindo o valor armazenado na variável `total` pelo valor armazenado na variável `cont`, por isso essa variável é do tipo

real, pois o número resultante da divisão poderá possuir casas decimais, o tipo inteiro não permite casas decimais.

Após a declaração dessas variáveis, inicia-se a execução do algoritmo atribuindo o número zero às variáveis `total` e `cont`. O objetivo é garantir que essas variáveis não contenham valores armazenados na memória do computador utilizados em outros programas.

O próximo passo consiste em solicitar que o usuário digite um valor para ser atribuído à variável `idade`. Após a variável `idade` receber o valor digitado pelo usuário, o algoritmo verificará a condição inicial do teste do laço, ou seja, se a idade digitada é maior que zero, pois a condição de parada da execução do laço é a idade ser menor ou igual a zero. Caso essa idade seja maior que zero, o resultado do teste da condição será verdadeiro e o algoritmo executará os comandos internos ao laço.

Os comandos iniciais internos do laço são: adicionar a idade digitada à variável `total` e somar 1 à variável `cont`. Essas variáveis são chamadas de acumuladores, pois em todas as iterações elas somarão novos valores aos valores já armazenados. O comando interno seguinte consiste em solicitar que o usuário digite uma nova idade.

Após o usuário digitar a nova idade, o ciclo do laço se repete, ou seja, o algoritmo testará a idade, se ela for maior que zero, executará os comandos internos do laço, essa repetição só parará quando uma idade menor ou igual a zero for digitada pelo usuário.

Quando o usuário digitar uma idade menor ou igual a zero, e execução do algoritmo sairá do laço, a média das idades será calculada e armazenada na variável `media`, esse valor é calculado dividindo o valor armazenado na variável `total` pelo valor armazenado na variável `cont`. Finalmente, o conteúdo da variável `media` será apresentado no monitor do computador ao usuário.

A seguir, será apresentada a execução do algoritmo implementado para o caso de o usuário digitar três idades válidas e uma idade negativa para o cálculo da média das idades. Repare que inicialmente não se sabe quantas idades o usuário digitará, o critério para a saída do laço e cálculo da média das idades é a digitação de uma idade menor ou igual a zero.

A figura 8.21 apresenta a declaração das variáveis pelo algoritmo.

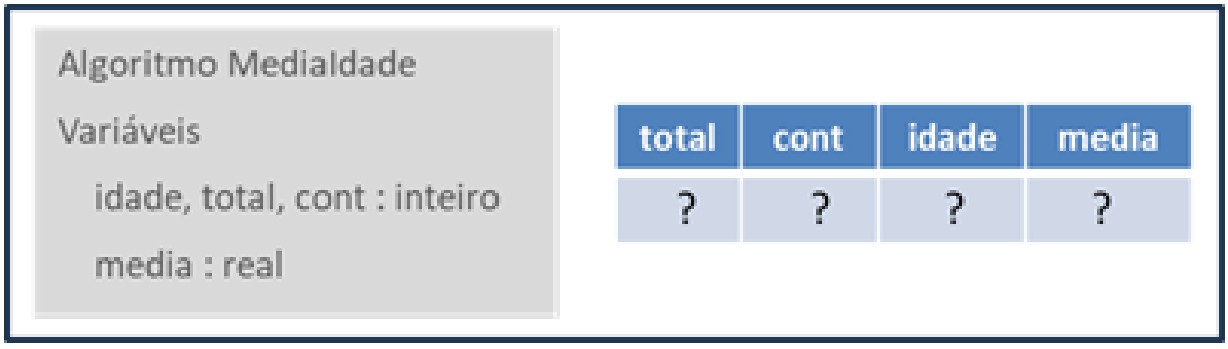


Figura 8.21: Declaração das variáveis.

O lado esquerdo da figura 8.21 apresenta a parte do pseudocódigo responsável pela declaração das variáveis que serão utilizadas. Neste caso, foram declaradas as variáveis `idade`, `total`, `cont`, todas do tipo inteiro, e a variável `media`, do tipo real. A tabela do lado direto dessa figura representa a memória do computador, pois uma variável é uma referência a uma área de memória do computador. Todas as variáveis possuem uma interrogação como conteúdo, essa interrogação representa a incerteza do conteúdo da área de memória referenciada pela variável, pois essa área de memória pode ter sido utilizada por outro programa e pode ter ficado com algum valor que para o pseudocódigo atual é desconhecido.

A figura 8.22 apresenta a inicialização de variáveis pelo algoritmo.

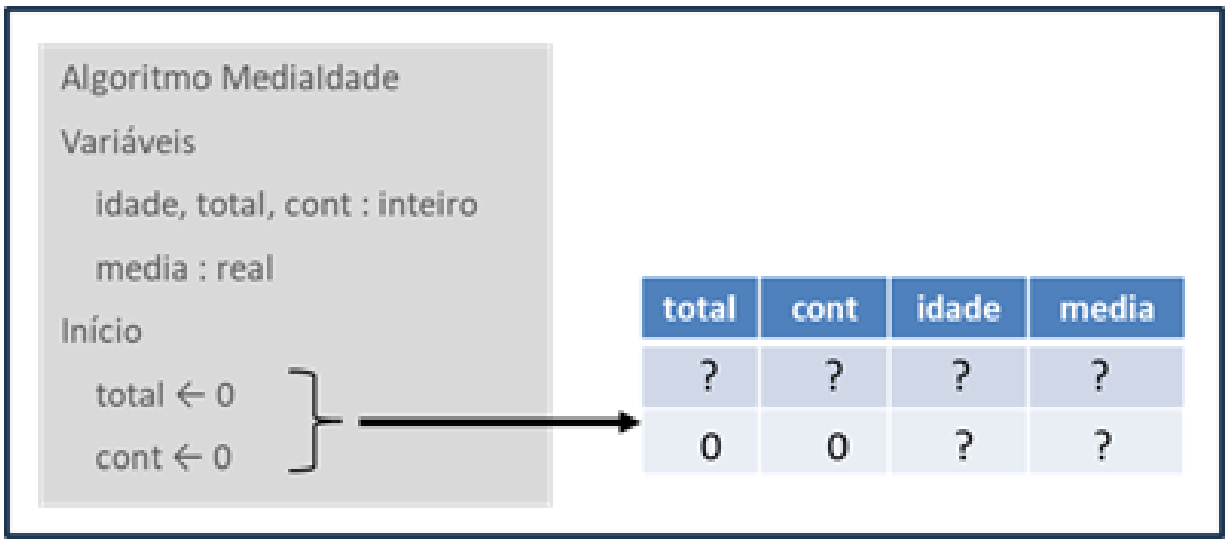


Figura 8.22: Inicialização das variáveis.

As duas linhas destacadas por uma chave do lado esquerdo da figura 8.22 apresentam a parte do pseudocódigo responsável pela inicialização das variáveis `total` e `cont`. Neste caso, elas foram inicializadas com o valor zero. A tabela do lado direto dessa figura representa a memória do computador, agora as variáveis `total` e `cont` armazenam o valor zero, as demais continuam com valores desconhecidos. O próximo passo será a leitura da primeira idade. A figura 8.23 apresenta a leitura da primeira idade.

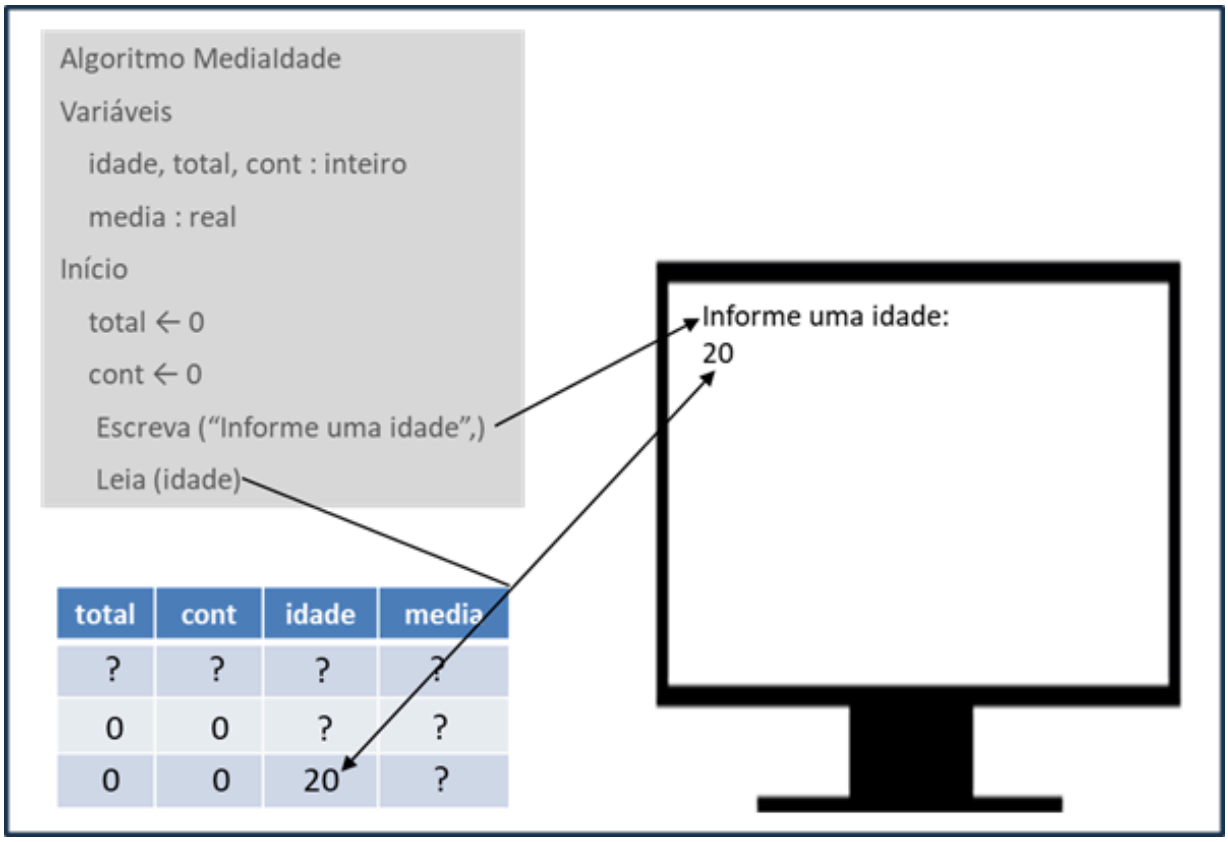


Figura 8.23: Leitura da primeira idade.

A linha do pseudocódigo com o comando `Escreva` apresentará no monitor do computador uma mensagem solicitando a digitação de uma idade. Após o usuário digitar uma idade, neste caso 20, ele deve pressionar a tecla `Enter` para que o comando `Leia` copie o número digitado para a área de memória referenciada pela variável `idade` . É possível observar na tabela na parte inferior do lado esquerdo da figura 8.23 o estado da memória após esse comando. O próximo passo será testar a idade digitada para a execução dos comandos internos ao laço, conforme apresentado na figura 8.24.

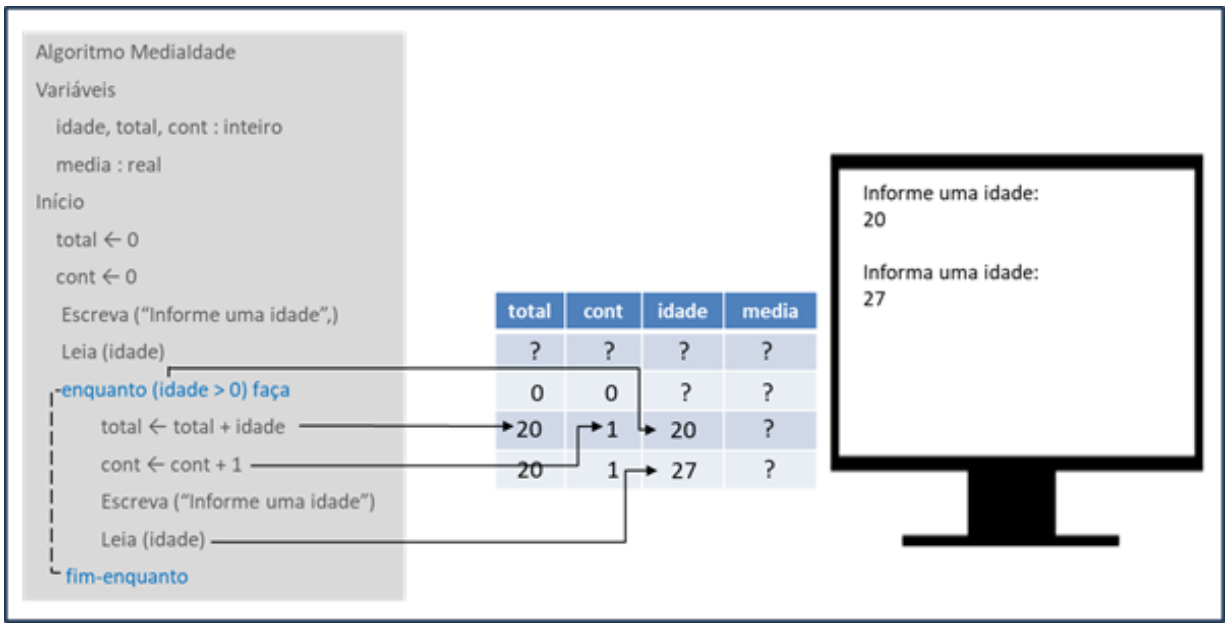


Figura 8.24: Primeira execução do Laço condicional com teste no início.

Conforme apresentado na figura 8.24, o teste para executar os comandos internos ao laço consiste em verificar se a variável `idade` é maior que zero, neste caso, como a variável `idade` armazena o valor 20, o laço será executado.

O primeiro comando do laço é somar à variável `total` o conteúdo da variável `idade`, então essa variável passará a valer 20 (0 da variável `total` + 20 da variável `idade`). O segundo comando é somar 1 à variável `cont`, então essa variável passará a valer 1. O próximo comando é o comando `Escreva` que apresentará no monitor do computador uma mensagem solicitando a digitação de uma idade. Após o usuário digitar uma idade, neste caso 27, ele deve pressionar a tecla `Enter` para que o comando `Leia` copie o número digitado para a área de memória referenciada pela variável `idade`. É possível observar na figura 8.24 o estado da memória após esse comando. O próximo passo será testar a idade digitada para a execução dos comandos internos ao laço, conforme apresentado na figura 8.25.

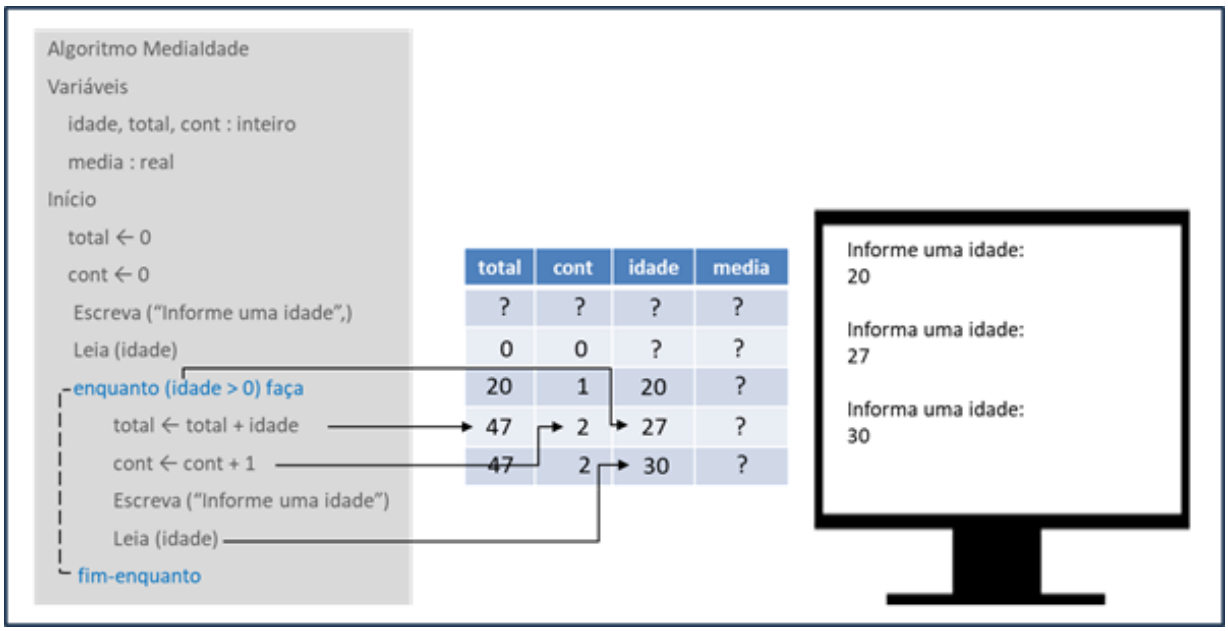


Figura 8.25: Segunda execução do Laço condicional com teste no início.

Conforme apresentado na figura 8.25, o teste para executar os comandos internos ao laço consiste em verificar se a variável `idade` é maior que zero, neste caso, como a variável `idade` armazena o valor 27, o laço será executado.

O primeiro comando do laço é somar à variável `total` o conteúdo da variável `idade`, então essa variável passará a valer 47 (20 da variável `total` + 27 da variável `idade`). O segundo comando é somar 1 à variável `cont`, então essa variável passará a valer 2. O próximo comando é o comando `Escreva` que apresentará no monitor do computador uma mensagem solicitando a digitação de uma idade. Após o usuário digitar uma idade, neste caso 30, ele deve pressionar a tecla `Enter` para que o comando `Leia` copie o número digitado para a área de memória referenciada pela variável `idade`. É possível observar na figura 8.25 o estado da memória após esse comando. O próximo passo será testar a idade digitada para a execução dos comandos internos ao laço, conforme apresentado na figura 8.26.

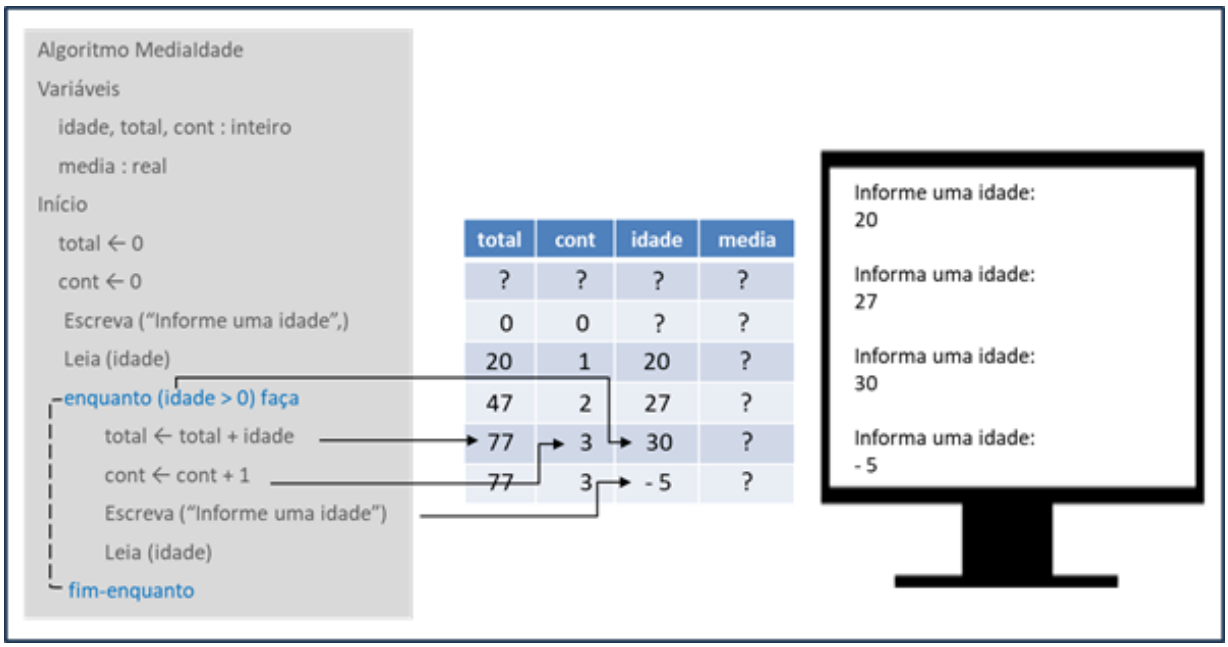


Figura 8.26: Terceira execução do Laço condicional com teste no início.

Conforme apresentado na figura 8.26, o teste para executar os comandos internos ao laço consiste em verificar se a variável `idade` é maior que zero, neste caso, como a variável `idade` armazena o valor 30, o laço será executado.

O primeiro comando do laço é somar à variável `total` o conteúdo da variável `idade`, então essa variável passará a valer 77 (47 da variável `total` + 30 da variável `idade`). O segundo comando é somar 1 à variável `cont`, então essa variável passará a valer 3. O próximo comando é o comando `Escreva` que apresentará no monitor do computador uma mensagem solicitando a digitação de uma idade. Após o usuário digitar uma idade, neste caso -5, ele deve pressionar a tecla `Enter` para que o comando `Leia` copie o número digitado para a área de memória referenciada pela variável `idade`. É possível observar na figura 8.26 o estado da memória após esse comando. O próximo passo será testar a idade digitada para a execução dos comandos internos ao laço, conforme apresentado na figura 8.27.

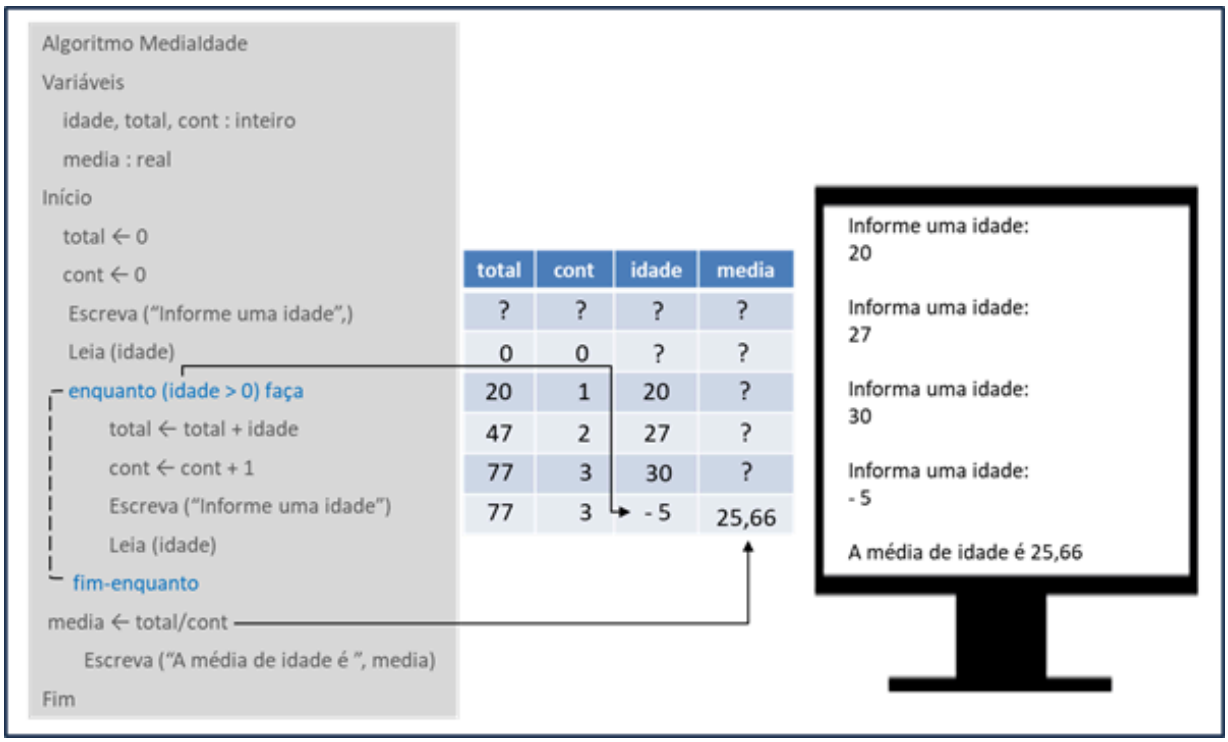


Figura 8.27: Quarta execução do Laço condicional com teste no início.

Conforme apresentado na figura 8.27, o teste para executar os comandos internos ao laço consiste em verificar se variável `idade` é maior que zero, neste caso, como a variável `idade` armazena o valor -5, ou seja, é menor que zero, o laço não será executado.

O primeiro comando fora do laço é armazenar a média das idades na variável `media`. A média das idades é calculada dividindo o conteúdo da variável `total` pelo conteúdo da variável `cont` que resultará em 77/3 cujo resultado será 25,66. O próximo comando é o comando `Escreva` que apresentará no monitor do computador a mensagem "A média de idade é" seguida pelo conteúdo da variável `media`, que neste caso é 25,66. Após esse passo, a execução do algoritmo será finalizada.

Laço condicional com teste no fim

O laço condicional com teste no fim executa um bloco de comandos para depois verificar se uma condição é verdadeira. Assim como no

laço condicional com teste no início, não é preciso saber, de antemão, quantas vezes será necessário repetir um conjunto de comandos em um algoritmo. A diferença do laço condicional com teste no fim para o laço condicional com teste no início é que o primeiro executa um bloco de comando para depois testar uma condição, já o segundo testa a condição para depois executar um bloco de comandos.

Esse laço repete a execução desse bloco enquanto a condição for verdadeira, quando a condição se torna falsa, execução do bloco é encerrada. A figura 8.28 apresenta um exemplo de fluxograma de um laço condicional com teste no fim com o pseudocódigo equivalente ao lado.

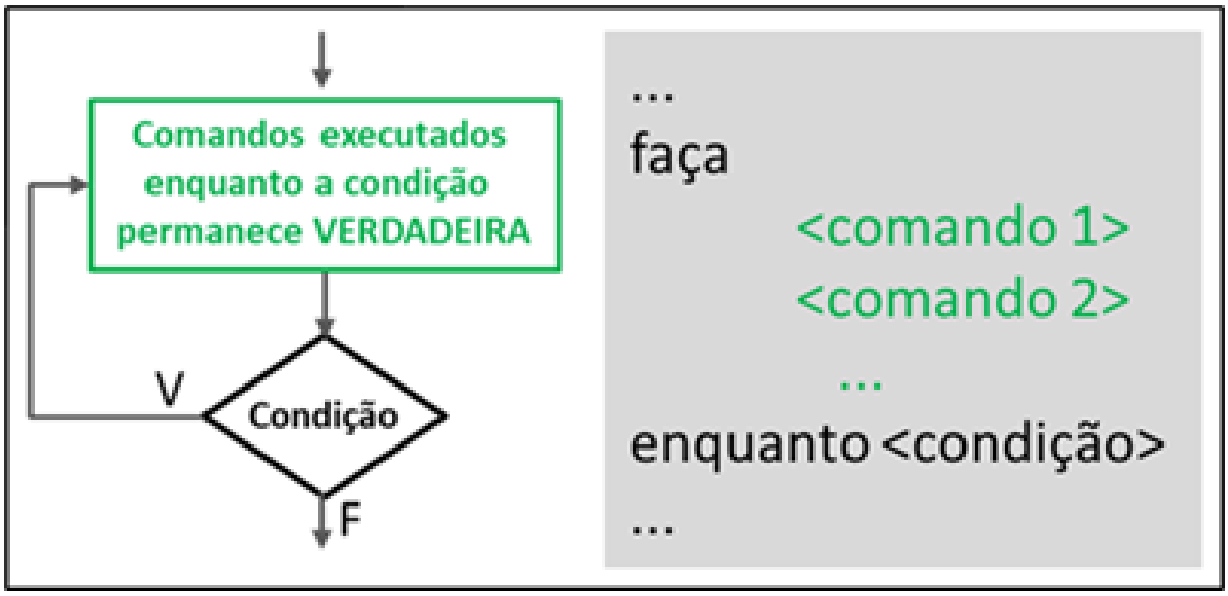


Figura 8.28: Laço condicional com teste no fim.

Como pode ser observado no fluxograma apresentado na figura 8.28, a estrutura de laço condicional com teste no fim é composta por um conjunto de comandos que serão executados várias vezes e pelo teste de uma condição no seu fim, cada execução desse conjunto de comandos caracteriza uma iteração. O término da execução acontece quando a condição testada se torna falsa.

À direita da figura 8.28, é apresentado o pseudocódigo equivalente ao fluxograma. Nele, o termo `faça` caracteriza o início do bloco de comandos a serem executados e a condição a ser testada compõe o termo `enquanto <condição>`.

Quando a execução do algoritmo atinge a linha `enquanto <condição>`, a condição é testada, caso o resultado do teste seja verdadeiro, o algoritmo volta para executar os comandos internos ao laço que são iniciados imediatamente abaixo da linha `faça`.

Após cada execução do bloco de comandos, o algoritmo testa se a condição é verdadeira, caso positivo, os comandos internos ao laço são executados novamente. Por outro lado, caso a condição testada seja falsa, o algoritmo sai do laço e executa os comandos externos e finaliza a execução do algoritmo.

Exemplo de algoritmo com laço condicional com teste no fim

Enunciado: implemente um algoritmo utilizando laço condicional com teste no fim que leia a idade de várias pessoas e no final apresente a média das idades. O critério de parada será a digitação de uma idade igual a zero ou negativa, ou seja, uma idade menor ou igual a zero.

A figura 8.29 apresenta o fluxograma e o pseudocódigo que implementam a solução do enunciado anterior utilizando laço condicional com teste no fim.

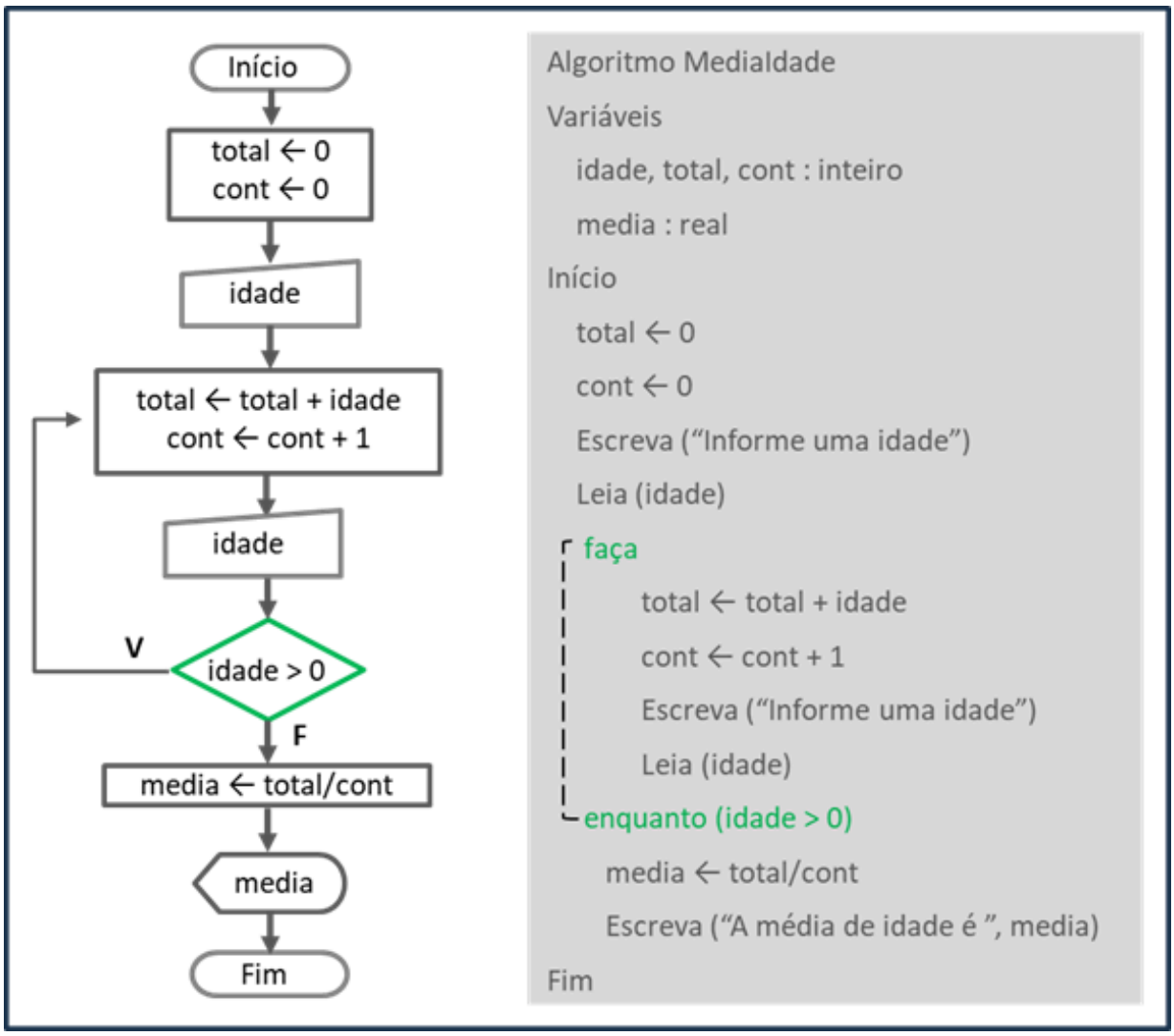


Figura 8.29: Exemplo de algoritmo com laço condicional com teste no fim.

Como pode ser observado no fluxograma e no pseudocódigo da figura 8.29, inicialmente são declaradas as variáveis `idade`, `total`, `cont`, todas do tipo inteiro, e a variável `media`, do tipo real. A variável `idade` armazenará cada idade digitada pelo usuário, a variável `total` armazenará a soma de todas as idades digitadas, a variável `cont` armazenará a quantidade de idades digitadas e a variável `média` armazenará a média das idades digitadas. Esse valor será obtido dividindo o valor armazenado na variável `total` pelo valor armazenado na variável `cont`, por isso essa variável é do tipo real, pois o número resultante da divisão poderá possuir casas

decimais, e o tipo inteiro não permite casas decimais. Após a declaração dessas variáveis, inicia-se a execução do algoritmo atribuindo o número zero às variáveis `total` e `cont`. O objetivo é garantir que essas variáveis não contenham valores armazenados na memória do computador utilizados em outros programas.

O próximo passo consiste em solicitar que o usuário digite um valor para ser atribuído à variável `idade`. Após esse passo, o algoritmo executará os comandos internos ao laço. Os comandos iniciais internos do laço são adicionar a idade digitada à variável `total` e somar um à variável `cont`. Essas variáveis são chamadas de acumuladores, pois em todas as interações elas somarão novos valores a valores já armazenados. O comando interno seguinte consiste em solicitar que o usuário digite uma nova idade.

Após a variável `idade` receber o valor digitado pelo usuário, o algoritmo verificará a condição do teste do laço, ou seja, se a idade digitada é maior que zero, pois a condição de parada da execução do laço é a idade ser menor ou igual a zero. Caso essa idade seja maior que zero, o resultado do teste da condição será verdadeiro e o algoritmo voltará e executará os comandos internos ao laço.

Quando o usuário digitar uma idade menor ou igual a zero, e execução do algoritmo sairá do laço, a média das idades será calculada e armazenada na variável `media`, esse valor é calculado dividindo o valor armazenado na variável `total` pelo valor armazenado na variável `cont`. Finalmente, o conteúdo da variável `media` será apresentado no monitor do computador ao usuário.

Após a apresentação de alguns recursos e exemplos de fluxograma e pseudocódigo, finalizamos o aprendizado sobre algoritmos. No entanto, cabe ressaltar que existem vários recursos utilizados para a implementação de algoritmos além dos apresentados neste livro, porém optamos por apresentar apenas um resumo sobre o assunto a fim de que você possa ler, entender e implementar algoritmos básicos.

No próximo capítulo apresentaremos a implementação dos algoritmos abordados neste capítulo utilizando a linguagem de programação Python para que você entenda os fundamentos de uma linguagem de programação.

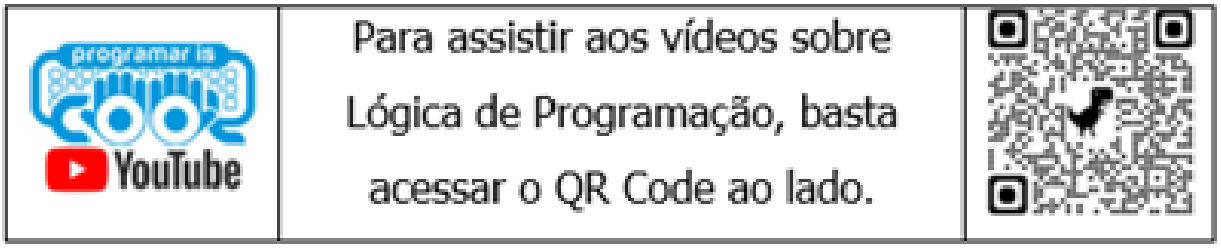


Figura 8.30: Vídeo sobre Lógica de Programação. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIDgCIGfffEruW8FvlN1GxgI

CAPÍTULO 9
Linguagem de programação Python

Segundo Aristóteles "*É fazendo que se aprende a fazer aquilo que se deve aprender a fazer*". Sendo assim, uma maneira de aprender a fazer o conteúdo apresentado no capítulo sobre lógica de programação é utilizar uma linguagem de programação para implementar os algoritmos abordados.

Neste capítulo, apresentaremos os fundamentos da linguagem de programação Python para que os leitores interessados em pôr em prática o conteúdo abordado no capítulo anterior conheçam uma ferramenta necessária para atingir esse objetivo. Ressalto novamente que este livro não possui o objetivo de formar programadores, por isso, neste capítulo, mostrarei apenas os recursos da linguagem de programação Python necessários para entender e implementar programas básicos.

Por que devo conhecer uma linguagem de programação?

Este livro não existiria se a computação não tivesse evoluído ao ponto de as máquinas serem capazes de processar o grande volume de informações disponibilizado nos mais diversos formatos no ambiente corporativo atual, pois foi nesse contexto que surgiram as organizações orientadas a dados e, consequentemente, a necessidade de que profissionais de qualquer área de formação sejam capazes de compreender, organizar e interpretar os dados para embasar as suas decisões, portanto, sejam alfabetizados em dados.

Embora, nas organizações orientadas a dados, as pessoas decisoras precisem interagir constantemente com computadores para analisar as informações úteis ao apoio à decisão, essas máquinas, nativamente, não entendem a linguagem natural (usada pelos seres humanos). Mesmo havendo um grande esforço na área de

processamento de linguagem natural, algumas atividades ainda exigem que o ser humano aprenda uma linguagem de programação a fim de proporcionar a comunicação efetiva entre homem/máquina.

Uma linguagem de programação proporciona a comunicação entre hardware (máquina) e o programador. É uma linguagem formal que funciona por meio de uma série de instruções, símbolos, palavras-chave, regras semânticas e sintáticas. Em programação, as regras semânticas focam no significado do conteúdo, enquanto as regras sintáticas especificam a forma de escrever o código-fonte (palavras reservadas, comandos, recursos diversos).

Então, mesmo não sendo necessário ser programador para ser considerado alfabetizado em dados, decidi incluir neste livro uma breve introdução a uma linguagem de programação para que você desenvolva a capacidade de ler e entender códigos de programação básicos.

Na próxima seção, abordaremos a origem da linguagem de programação Python e os fundamentos da sua sintaxe. Essa linguagem foi escolhida por ser uma das mais fáceis de se aprender e uma das mais utilizadas em análise de dados. Assim, o leitor ou leitora que se interessar em aprofundar seu aprendizado em análise de dados já terá sido apresentado(a) a uma das principais linguagens de programação utilizadas para desenvolver códigos com esse objetivo.

9.1 Origem da linguagem de programação Python

A linguagem Python foi criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI). Ela tinha originalmente foco em usuários como físicos e engenheiros, e foi concebida a partir de

outra linguagem existente na época, chamada ABC. Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation (PSF), é escrita na linguagem C e, por isso, é também conhecida como CPython. O nome Python foi tirado por Guido van Rossum do programa humorístico da TV britânica *Monty Python Flying Circus*.

Python é um exemplo de linguagem de programação de alto nível, isso significa que, apesar de ser uma linguagem peculiar a computadores, ela é semelhante à linguagem natural. Por outro lado, as linguagens de baixo nível, às vezes chamadas de linguagens de máquina, são as linguagens possíveis de serem processadas pelos computadores.

Sendo assim, os programas escritos em linguagem de alto nível precisam ser traduzidos para linguagem de baixo nível antes que possam ser executados pelos computadores.

Existem três estratégias para essa tradução. A primeira consiste em compilar o programa fonte, escrito na linguagem de alto nível, para um programa executável, gerado em linguagem de máquina, que pode ser executado pelo computador. A segunda consiste em usar um programa interpretador para traduzir cada comando do código-fonte para um comando de baixo nível executável pelo computador. A terceira consiste em gerar um código intermediário entre o código-fonte e o executável e usar um interpretador para traduzir cada comando desse código intermediário para um comando de baixo nível executável pelo computador.

A linguagem Python usa a terceira estratégia. A figura 9.1 apresenta um exemplo de ambiente de execução de programas escritos na linguagem de programação Python.

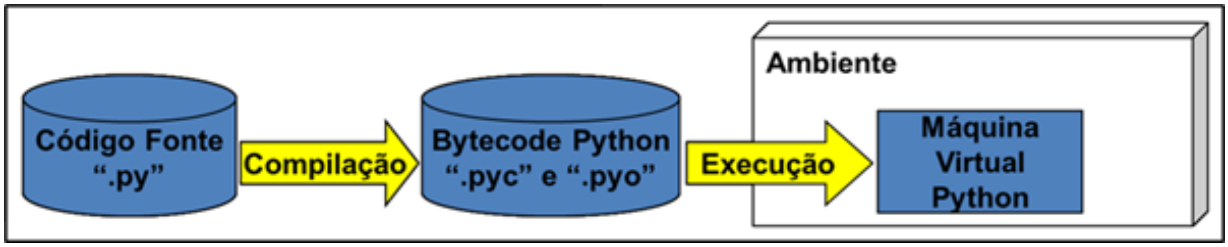


Figura 9.1: Exemplo de ambiente de execução de programas escritos em Python.

Como pode ser observado na figura 9.1, inicialmente o programador escreve um código-fonte utilizando a linguagem Python, gerando um arquivo com extensão `.py`. Esse arquivo é compilado para gerar um bytecode que pode ser um arquivo com extensão `.pyc` ou `pyo`.

O bytecode é uma forma de representação intermediária de um programa de computador. Ele é gerado a partir do código-fonte original e é projetado para ser executado por uma máquina virtual específica. Essa máquina virtual é um tipo de interpretador instalado no sistema operacional do computador no qual o bytecode será executado.

Essa estratégia tem como vantagem oferecer portabilidade do bytecode, permitindo que os programas sejam executados em diferentes sistemas operacionais e arquiteturas de hardware, desde que possuam uma máquina virtual Python instalada.

9.2 Sintaxe da linguagem de programação Python

A sintaxe de uma linguagem de programação compreende o conjunto de regras que determina quais combinações de símbolos e palavras-chaves podem ser utilizados para escrever qualquer instrução. Na verdade, é um tipo de gramática, sendo assim,

estabelece quais palavras, símbolos e operadores compõem a linguagem e em que ordem devem ser escritos.

A seguir serão apresentados os principais componentes da sintaxe Python.

Comentários

O caractere `#` marca o início de um comentário que ocupe apenas uma linha. Qualquer texto depois do `#` será ignorado até o fim da linha. Exemplo:

```
# Uma linha de comentário
```

O comentário contendo várias linhas é iniciado e fechado com a sequência de 3 aspas duplas `"""`. Exemplo:

```
"""
Primeira linha de comentário ...
Segunda linha de comentário ...
Terceira linha de comentário...
etc...
"""
```

Nomes de variáveis

Programadores geralmente escolhem nomes significativos para suas variáveis, pois os nomes podem indicar para o que a variável é usada. Os nomes das variáveis podem conter tanto letras quanto números, mas têm de começar com uma letra. Letras maiúsculas e minúsculas são diferentes, então `Nota` e `nota` são variáveis distintas. O caractere *underline* (*underscore* ou subtraço) `_` pode aparecer em um nome de variável. Ele é muito utilizado em nomes com múltiplas palavras, por exemplo, poderíamos ter em um programa as variáveis `nota_aluno` e `primeiro_nome_aluno`.

Tipos de variáveis

Os principais tipos de variáveis da linguagem Python são:

- **Integer:** inteiro, com abreviação `int`. É utilizado para representar números inteiros, sejam eles positivos ou negativos. Por exemplo: `10` ou `- 25`.
- **Floating-point:** número de ponto flutuante, com abreviação `float`. É utilizado para representar números reais que possuem uma parte decimal. Por exemplo: `15,25` ou `- 35,47`.
- **String:** cadeia de caracteres, com abreviação `str`. Consiste em sequência de caracteres e deve estar sempre entre aspas simples ou duplas. É utilizado para representar literais. Por exemplo: `'Bom dia!'`.
- **Boolean:** booleano, com abreviação `bool`. Representam os valores lógicos `True` (verdadeiro) e `False` (falso).

Cabe ressaltar que a linguagem de programação Python disponibiliza vários outros tipos de dados, porém os apresentados anteriormente são suficientes para atingirmos o objetivo deste capítulo.

Operadores Aritméticos

- **Soma (`+`)**
- **Diferença (`-`)**
- **Multiplicação (`*`)**
- **Divisão (`/`)**
- **Divisão inteira (`//`):** o resultado é truncado para o inteiro imediatamente inferior ao resultado da divisão. Por exemplo: `5 // 3 = 1` (a parte decimal do resultado é ignorada).
- **Módulo (`%`):** retorna o resto da divisão entre dois números. Por exemplo: `5 % 3 = 2` (5 / 3 = 1 e o resto é 2).
- **Potência (`**`):** eleva um número a uma potência. Por exemplo `5**2` (cinco ao quadrado).

Operadores Relacionais

- **Menor (<)**
- **Maior (>)**
- **Menor ou igual (<=)**
- **Maior ou igual (>=)**
- **Igual (==)**
- **Diferente (!=)**

Operadores Lógicos

- **and:** retorna um valor verdadeiro se, e somente se, receber duas expressões que forem verdadeiras.
- **or:** retorna um valor verdadeiro se pelo menos uma das condições for verdadeira.
- **not:** inverte o valor da condição, retornando verdadeiro se a condição for falsa e vice-versa.

Após a breve introdução à sintaxe Python mostrada anteriormente, a seguir, será exibida a implementação de um algoritmo bem simples para apresentar um exemplo de código implementado em Python.

Exemplo de código em Python

O algoritmo escolhido calcula a soma de dois números, a sua implementação utilizando pseudocódigo já foi detalhada na figura 8.4 do capítulo anterior, porém apresentaremos novamente para comparar esse pseudocódigo com a codificação em Python. A figura 9.2 apresenta esse algoritmo implementado em pseudocódigo (à esquerda) e em Python (à direita).

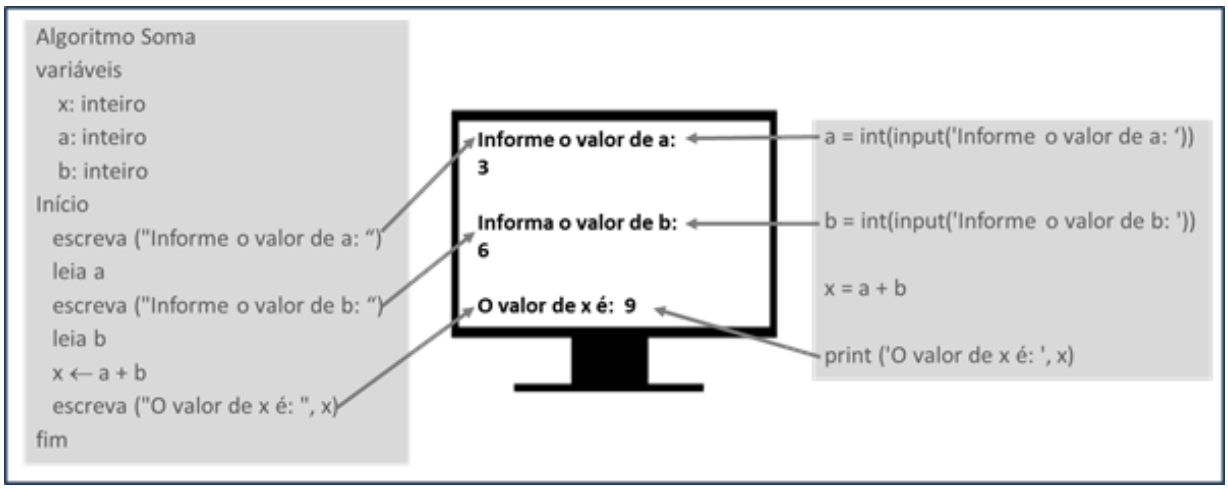


Figura 9.2: Algoritmo implementado em Python.

```
a = int(input('Informe o valor de a:  '))
b = int(input('Informe o valor de b:  '))

x = a + b

print ('O valor de x é: ', x)
```

A figura 9.2 apresenta algoritmo que calcula a soma de dois números implementado em pseudocódigo e em Python. Como pode ser observado, é muito mais simples implementar um algoritmo em Python (à direita) do que em pseudocódigo (à esquerda).

O pseudocódigo, inicialmente, declara as variáveis `x`, `a` e `b`, que são do tipo inteiro. Em Python não é necessário declarar uma variável para depois atribuir um valor, podemos declarar a variável e, no mesmo comando, atribuir um valor.

Em seguida, o comando do pseudocódigo `escreva ("Informe o valor de a:")` exibe a mensagem "Informe o valor de a:" no monitor, instruindo o usuário a digitar um número. Esse comando é equivalente ao comando Python `a = int(input('Informe o valor de a:'))`, o comando `input` também exibe a mensagem "Informe o valor de a:" para o usuário e já prepara a variável `a` para receber o valor digitado pelo usuário. Por default Python considera a variável

lida como sendo uma `string` (literal), caso o programador deseje que essa variável seja de outro tipo, ele deve indicar no comando, neste caso, foi indicado que a variável deveria ser lida como um inteiro por meio da palavra `int` no início do comando.

Após o usuário digitar o número 3 e pressionar a tecla `Enter`, o pseudocódigo atribui esse valor à variável `a` por meio do comando `leia a`. Em Python, esse comando `leia a` já está incluído na linha `a = int(input('Informe o valor de a:'))`, não há a necessidade de um comando explícito para atribuir à variável `a` o valor digitado, tornando a linguagem Python mais simples. Os mesmos passos serão seguidos para atribuir o valor 6 à variável `b`.

O próximo passo consiste em atribuir o resultado da soma dos valores armazenados nas variáveis `a` e `b` à variável `x` por meio da linha `x <- a + b`, no pseudocódigo, ou `x = a + b` em Python. Após essa atribuição, o algoritmo exibe a mensagem "O valor de x é:" concatenada ao número armazenado na variável x por meio do comando em pseudocódigo `escreva ("O valor de x é: ", x)` e, em Python, `print ('O valor de x é: ', x)`.

Nas próximas seções serão apresentados os blocos de código utilizados para implementar decisões e repetições em programas escritos na linguagem Python.

9.3 Blocos de código Python

Um bloco é uma parte do conteúdo do programa Python que é executado como uma unidade. Os blocos de código são delimitados pelo uso de indentação. A linha imediatamente anterior ao início do bloco sempre termina com dois pontos `:` e representa uma estrutura de controle da linguagem ou uma declaração de uma nova

estrutura. A seguir serão apresentados os blocos de código mais comuns em Python.

Desvio Condicional Simples

O pseudocódigo relativo ao desvio condicional simples já foi apresentado em detalhes no capítulo anterior. Aqui apresentaremos o fluxograma e a sintaxe Python equivalente para a implementação desse desvio. A figura 9.3 apresenta o desvio condicional simples implementado em Python.

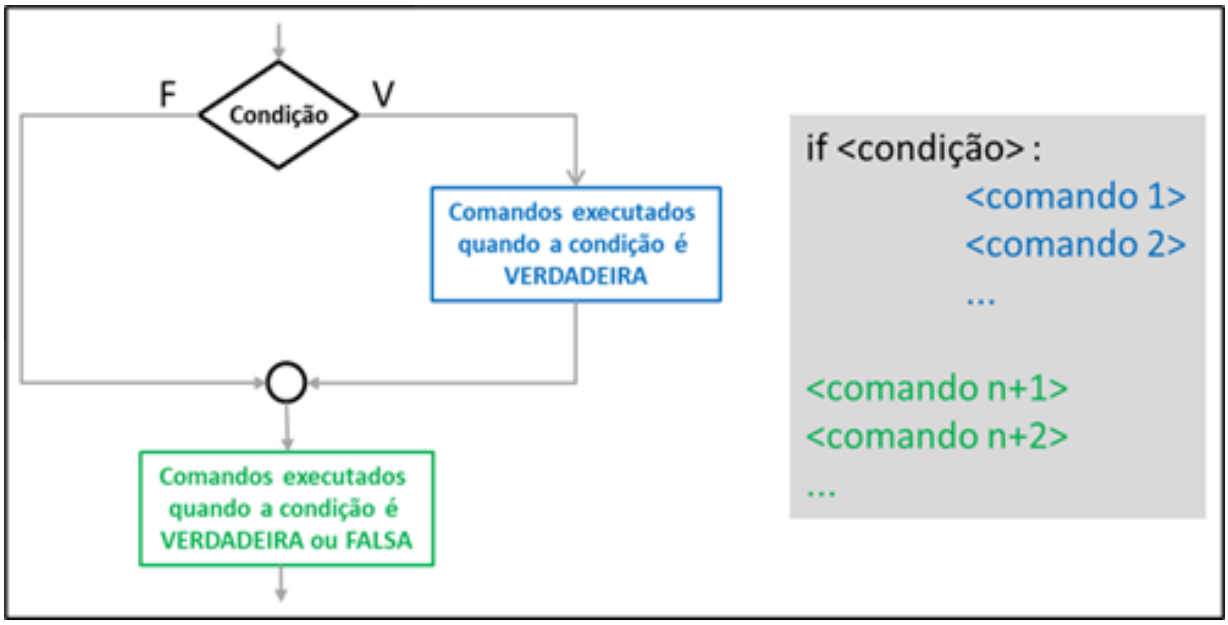


Figura 9.3: Desvio condicional simples implementado em Python.

À direita da figura 9.3, é apresentado o código Python equivalente ao fluxograma que representa o desvio condicional simples. No código, a condição a ser testada está no início do desvio representada pela linha `if <condição>:` . Os comados internos ao bloco estão indentados (equivalente ao espaçamento ao pressionar a tecla `Tab`) e o fim desse desvio condicional está representado pela ausência da indentação dos comandos, assim, os comandos internos

ao bloco iniciado por `if` estão indentados e os externos estão alinhados à palavra `if`.

Quando a execução do algoritmo atinge a linha `if <condição>:`, se a condição for verdadeira, o algoritmo executará os comandos indentados, imediatamente abaixo dessa instrução, e depois irá para o fim da estrutura de decisão. Após esse passo, o algoritmo executará os comandos sem indentação, consequentemente, os comandos alinhados à palavra `if`, pois eles estão fora da estrutura de decisão.

Por outro lado, se a condição for falsa, o algoritmo irá diretamente para os comandos sem indentação, pulando os comandos internos, que são os imediatamente abaixo da instrução `if <condição>:` indentados, pois os comandos sem indentação estão fora da estrutura de decisão.

Exemplo de programa com desvio condicional simples

Enunciado: implemente um programa em Python com um desvio condicional simples que receba dois números e teste se a soma desses dois números é maior que 10. Caso o resultado desse teste seja verdadeiro, o algoritmo deve exibir a soma, caso contrário, nenhum número será exibido. Quando finalizar o desvio condicional, o algoritmo deverá imprimir a frase "Estou fora da estrutura de decisão." antes de finalizar sua execução.

A figura 9.4 apresenta o pseudocódigo (à esquerda) e o código Python (à direita) que implementam a solução do enunciado anterior. Cabe ressaltar que esse pseudocódigo já foi explicado em detalhes no capítulo anterior e, aqui, daremos ênfase ao código Python.

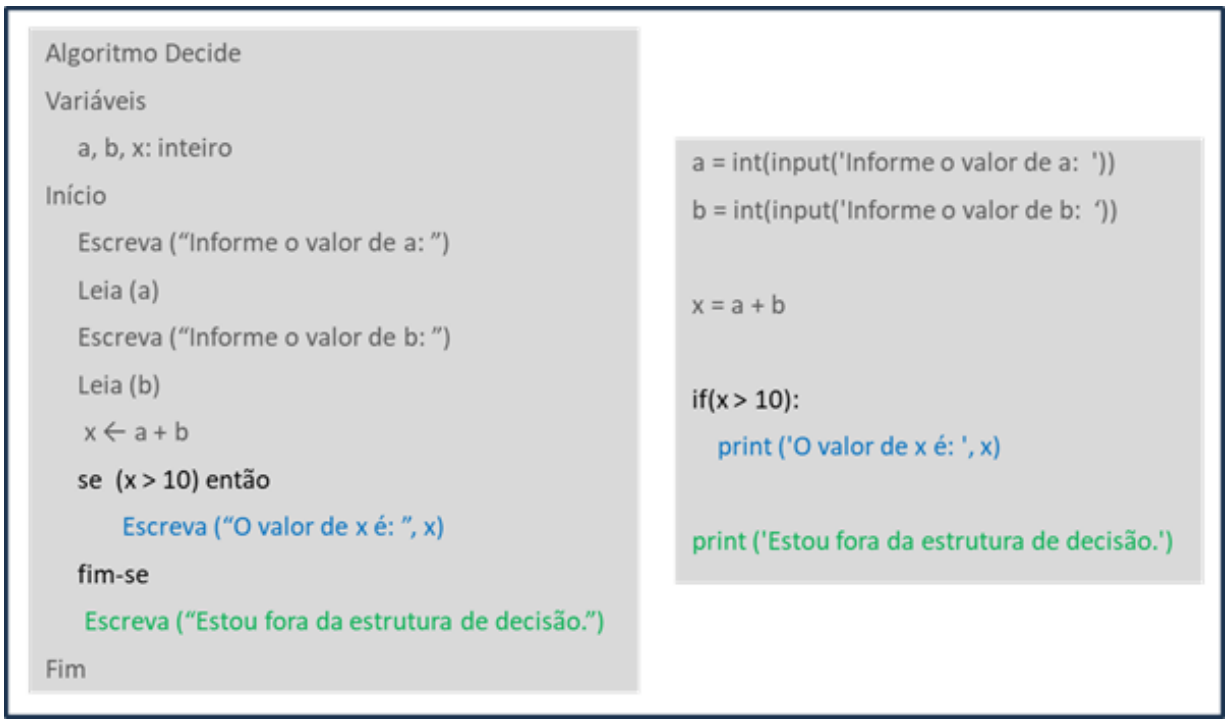


Figura 9.4: Exemplo de programa com desvio condicional simples.

```
a = int(input('Informe o valor de a:  '))
b = int(input('Informe o valor de b:  '))

x = a + b

if(x > 10):
    print ('O valor de x é: ', x)

print ('Estou fora da estrutura de decisão.')
```

Como pode ser observado na figura 9.4, o código Python inicia com o comando `a = int(input('Informe o valor de a:'))`, o comando `input` exibe a mensagem "Informe o valor de a:" para o usuário e já prepara a variável `a` para receber o valor digitado por ele. Por padrão Python considera a variável lida como sendo uma `string` (literal), caso o programador deseje que essa variável seja de outro tipo, ele deve indicar no comando. Neste caso, foi indicado que a variável deveria ser lida como um inteiro por meio da palavra `int` no início do comando.

Após o usuário digitar o valor desejado e pressionar a tecla `Enter`, a variável `a` recebe o valor digitado. Em Python, essa operação de atribuição já está incluída na linha `a = int(input('Informe o valor de a:'))`, não há a necessidade de um comando explícito para atribuir à variável `a` o valor digitado. Os mesmos passos serão seguidos para atribuir um valor à variável `b`.

O próximo passo consiste em atribuir o resultado da soma dos valores armazenados nas variáveis `a` e `b` à variável `x` por meio da linha `x = a + b`. Após essa atribuição, o algoritmo testa se `x` é maior que 10 na linha `if(x > 10):`, se o valor de `x` for maior que 10, o algoritmo executa os comandos internos à estrutura de decisão, portanto, os comandos indentados, neste caso, o comando `print ('O valor de x é: ', x)`. Esse comando exibe a mensagem "O valor de x é:" concatenada ao número armazenado na variável `x`.

Por outro lado, se o valor de `x` for menor ou igual a 10, a execução vai direto para o fim da estrutura de decisão sem executar os comandos indentados, neste caso a linha `print ('O valor de x é: ', x)`.

Após o fim da estrutura de decisão, o algoritmo exibirá a frase "Estou fora da estrutura de decisão.", ou seja, independentemente de o resultado do teste ser verdadeiro ou falso o algoritmo sempre apresentará a frase "Estou fora da estrutura de decisão.", pois o comando `print ('Estou fora da estrutura de decisão.')` encontra-se após o fim da estrutura de decisão, por não estar indentado. Por fim, a execução do algoritmo é finalizada.

Desvio Condicional Composto

O pseudocódigo relativo ao desvio condicional composto já foi apresentado em detalhes no capítulo 8, nesta seção apresentaremos o fluxograma e a sintaxe Python equivalente para a implementação

desse desvio. A figura 9.5 apresenta o desvio condicional composto implementado em Python.

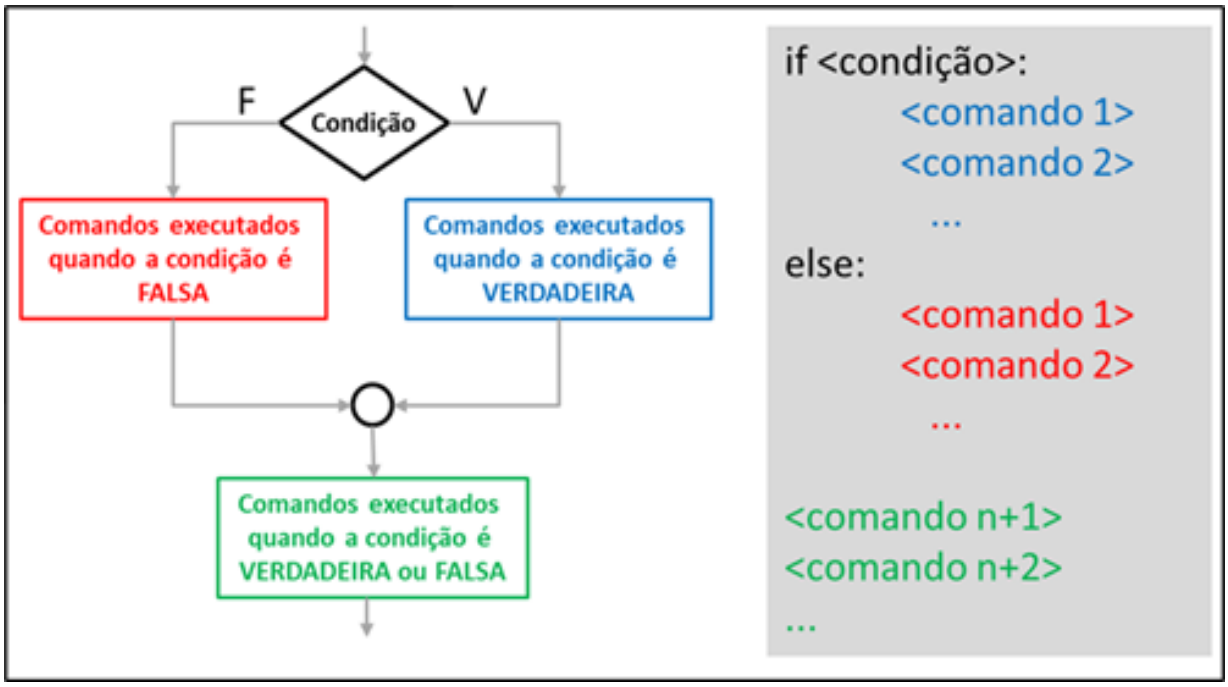


Figura 9.5: Desvio condicional composto implementado em Python.

À direita da figura 9.5, é apresentado o código Python equivalente ao fluxograma que representa o desvio condicional composto. No código, a condição a ser testada está no início do desvio representada pela linha `if <condição>:` . Os comados internos ao bloco estão indentados (equivalente ao espaçamento ao pressionar a tecla `Tab`) e o fim desse desvio condicional está representado pela ausência da indentação dos comandos, assim, os comandos internos ao bloco iniciado por `if` estão indentados e os externos estão alinhados à palavra `if` .

Quando a execução do algoritmo atinge a linha `if <condição>:` , se a condição for verdadeira, o algoritmo executará os comandos indentados, imediatamente abaixo dessa instrução, e depois irá para o fim da estrutura de decisão, sem executar os comandos indentados após alinha `else:` . Após esse passo, o algoritmo

executará os comandos sem indentação, portanto, os comandos alinhados à palavra `if`, pois eles estão fora da estrutura de decisão.

Por outro lado, se a condição for falsa, o algoritmo irá diretamente para a linha `else:` e executará os comandos indentados abaixo dessa linha, pulando os comandos indentados após alinha `if <condição>:`. Depois, o algoritmo executará os comandos fora da estrutura de decisão, isto é, os comandos sem indentação.

Exemplo de programa com desvio condicional composto

Enunciado: implemente um programa em Python com um desvio condicional composto que receba dois números e teste se a soma desses dois números é maior que 10. Caso o resultado desse teste seja verdadeiro, o algoritmo deve exibir o valor do primeiro número digitado, caso contrário, o algoritmo deve exibir o valor do segundo número digitado. Quando finalizar o desvio condicional, o algoritmo deverá imprimir a frase "Estou fora da estrutura de decisão." antes de finalizar sua execução.

A figura 9.6 apresenta o pseudocódigo (à esquerda) e o código Python (à direita) que implementam a solução do enunciado anterior, cabe ressaltar que esse pseudocódigo já foi explicado em detalhes e nesta seção daremos ênfase ao código Python.

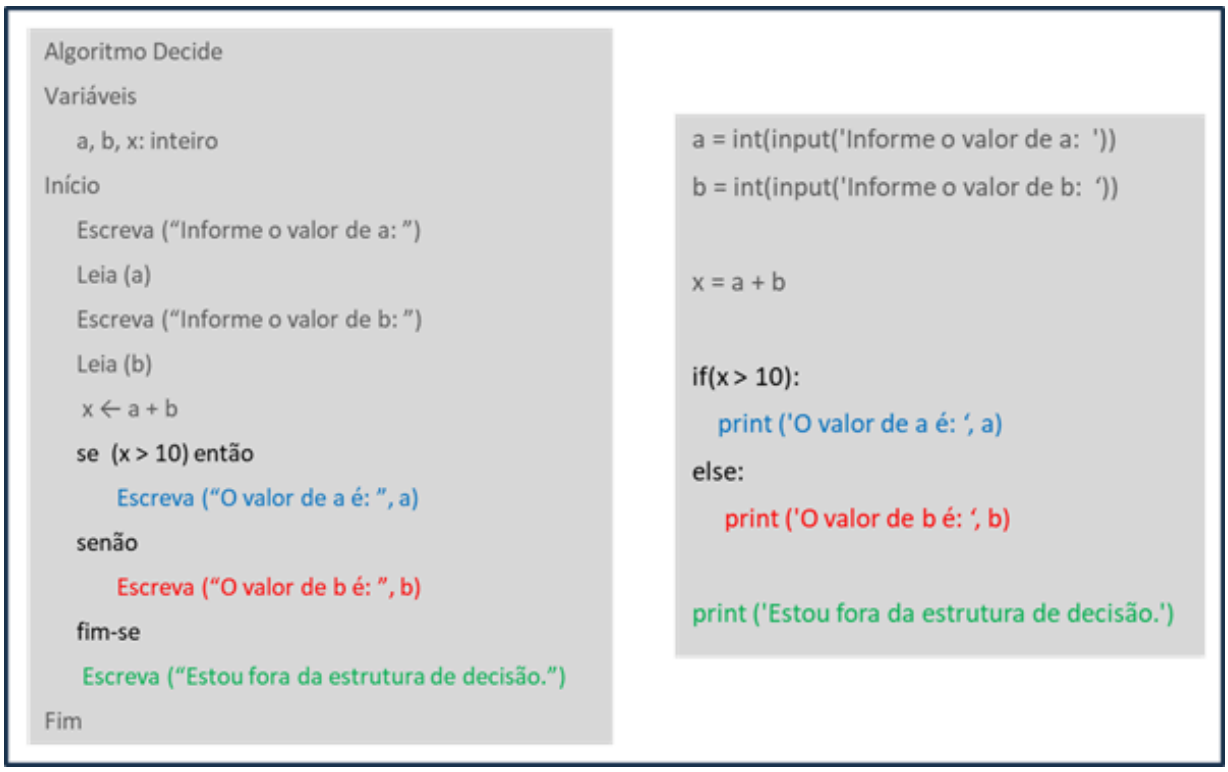


Figura 9.6: Exemplo de algoritmo com desvio condicional composto.

```
a = int(input('Informe o valor de a:  '))
b = int(input('Informe o valor de b:  '))

x = a + b

if(x > 10):
    print ('O valor de a é: ', a)
else:
     print ('O valor de b é: ', b)

print ('Estou fora da estrutura de decisão.')
```

Como pode ser observado na figura 9.6, o código Python inicia com o comando `a = int(input('Informe o valor de a:'))`, o comando `input` exibe a mensagem "Informe o valor de a:" para o usuário e já prepara a variável `a` para receber o número inteiro digitado por ele. Após o usuário digitar o valor desejado e pressionar a tecla `Enter`, a

variável `a` recebe o valor digitado. Os mesmos passos serão seguidos para atribuir um número inteiro à variável `b`.

O próximo passo consiste em atribuir o resultado da soma dos valores armazenados nas variáveis `a` e `b` à variável `x` por meio da linha `x = a + b`. Após essa atribuição, o algoritmo testa se `x` é maior que 10 na linha `if(x > 10):`, se o valor de `x` for maior que 10, o algoritmo executa os comandos indentados após a linha `if(x > 10):`, neste caso, o comando print `('O valor de a é: ', a)`, e depois irá para o fim da estrutura de decisão, sem executar os comandos indentados após alinha `else:`.

Por outro lado, se `x` for menor ou igual a 10, o algoritmo irá diretamente para a linha `else:` e executará os comandos indentados abaixo dessa linha, neste caso, o comando `print ('O valor de b é: ', b)`, pulando os comandos indentados após alinha `if <condição>:`.

Após o fim da estrutura de decisão, o algoritmo exibirá a frase "Estou fora da estrutura de decisão.", assim sendo, independentemente de o resultado do teste ser verdadeiro ou falso o algoritmo sempre apresentará essa frase, pois o comando `print ('Estou fora da estrutura de decisão.')` encontra-se após o fim da estrutura de decisão, por não estar indentado. Por fim, a execução do algoritmo é finalizada.

Laço Incondicional ou Contado

O pseudocódigo relativo ao laço incondicional ou contado já foi apresentado em detalhes no capítulo 8, nesta seção apresentaremos o fluxograma e a sintaxe Python equivalente para a implementação desse laço. A figura 9.7 apresenta o laço incondicional ou contado implementado em Python.

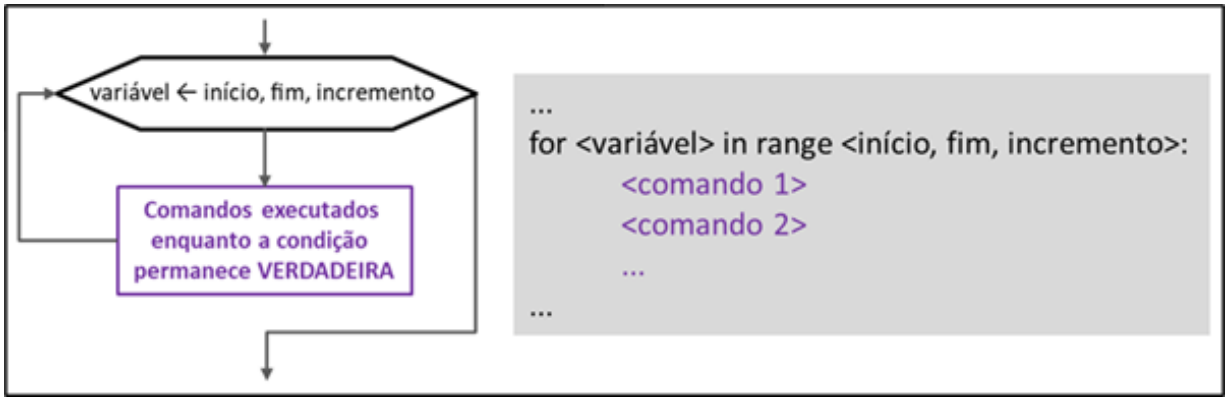


Figura 9.7: Laço incondicional ou contado implementado em Python.

À direita da figura 9.7, é apresentado o código Python equivalente ao fluxograma que representa o laço incondicional ou contado. Nele, a escolha da variável `<variável>`, a atribuição do valor inicial, o valor limite e o incremento `<início, fim, incremento>` são representados na linha `for <variável> in range <início, fim, incremento>:`. O conjunto de comandos que pertencem ao laço é limitado pela indentação.

Quando a execução do algoritmo atinge a linha `for <variável> in range <início, fim, incremento>:` pela primeira vez, a variável escolhida recebe o valor inicial e executa os comandos internos ao laço.

Após essa execução, a variável escolhida é incrementada pelo valor do incremento e o algoritmo testa se o valor da variável atingiu o valor limite. Caso não tenha atingido, os comandos internos ao laço são executados novamente e a variável é incrementada.

Por outro lado, caso o valor da variável tenha atingido o valor limite, o algoritmo sai do laço e executa os comandos externos ao laço, isto é, os comandos sem indentação, e finaliza a execução.

Exemplo de programa com laço incondicional ou contado

Enunciado: implemente um algoritmo utilizando laço incondicional ou contado que apresente no monitor do usuário os valores inteiros de 1 até 3.

A figura 9.8 apresenta o pseudocódigo (à esquerda) e o código Python (à direita) que implementam a solução do enunciado anterior, cabe ressaltar que esse pseudocódigo já foi explicado em detalhes e, aqui, daremos ênfase ao código Python.

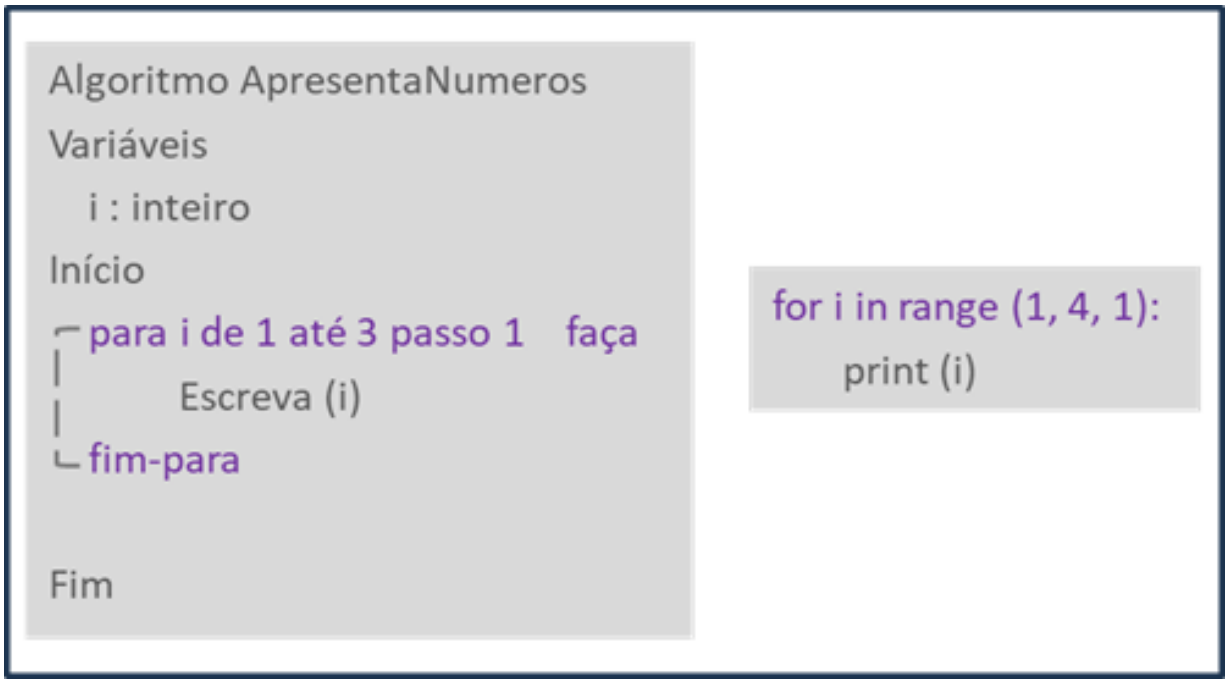


Figura 9.8: Exemplo de algoritmo com laço incondicional ou contado.

```
for i in range (1, 4, 1):
    print (i)
```

Como pode ser observado na figura 9.8, o código Python é muito simples, formado apenas por duas linhas. Na primeira linha, o comando `for i in range (1, 4, 1):`, atribui o número 1 à variável `i` por ser o valor inicial, o número 4 é estabelecido o limite e o número 1 é estabelecido como o incremento devido à parte do comando `(1, 4, 1)`.

Após a variável `i` receber o valor 1, o algoritmo executará o comando interno ao laço, `print (i)`, que consiste na exibição do valor armazenado na variável `i` no monitor do computador onde o programa está sendo executado.

Em seguida, devido ao fato de não haver mais comandos no interior do laço, a variável `i` é incrementada pelo valor 1 e volta para o teste do laço existente na primeira linha, para verificar se a variável `i` atingiu o valor 4. Caso o valor seja inferior a 4, o algoritmo executará o comando de exibição da variável `i` novamente; caso o valor seja igual ou superior a 4, o laço será encerrado e o algoritmo será finalizado. Assim, serão exibidos no monitor os valores 1, 2, 3, conforme a solicitação do enunciado.

Observe que há uma diferença entre o pseudocódigo e o programa em Python em relação à condição limite para executar os comandos internos ao laço. No pseudocódigo, quando o valor limite é atingido, os comandos internos ao laço são executados pela última vez. Já no programa em Python, quando o valor máximo é atingido os comandos internos não são mais executados, por isso, se for necessário imprimir os valores de 1 a 3, conforme o enunciado, é necessário colocar o valor máximo igual a 4, pois assim ele executará até a variável atingir o valor 3.

Laço Condicional com Teste no Início

O pseudocódigo relativo ao laço condicional com teste no início já foi apresentado em detalhes no capítulo 8, nesta seção apresentaremos o fluxograma e a sintaxe Python equivalente para a implementação desse laço. A figura 9.9 apresenta o laço condicional com teste no início implementado em Python. Cabe ressaltar que a linguagem de programação Python não disponibiliza a estrutura de laço condicional com teste no fim, no entanto, o laço condicional com teste no início supre essa lacuna.

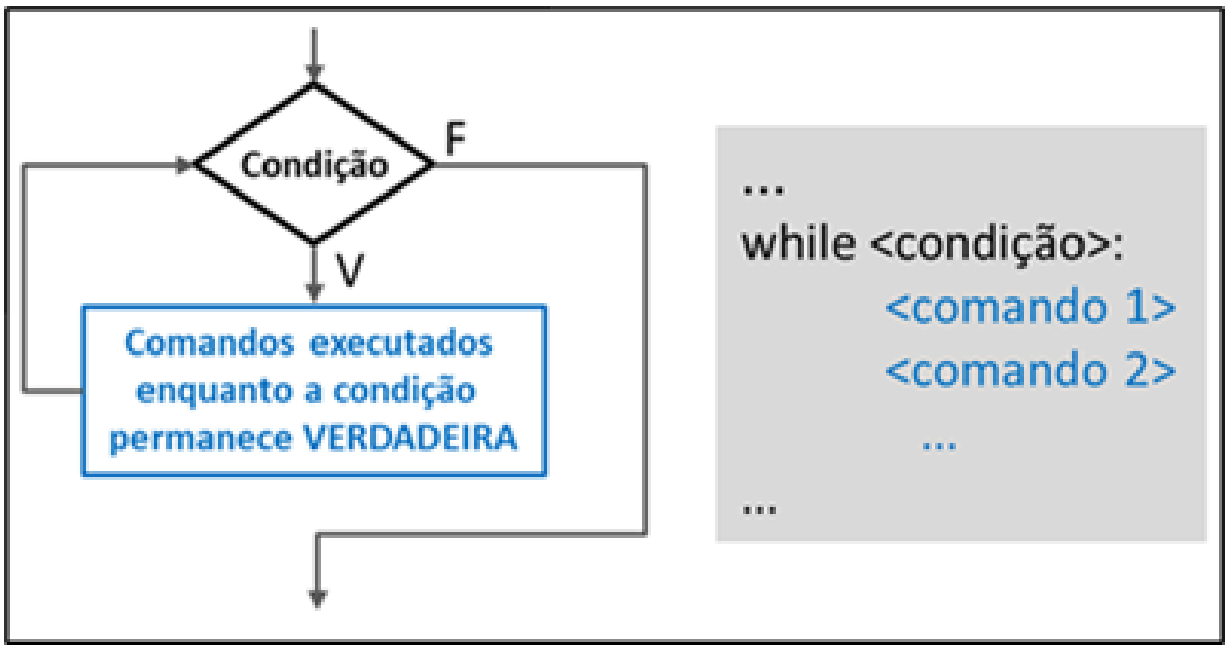


Figura 9.9: Laço condicional com teste no início implementado em Python.

À direita da figura 9.9, é apresentado o código Python equivalente ao fluxograma. Nele, a condição a ser testada compõe o termo `while <condição>:`. Quando a execução do algoritmo atinge essa linha, a condição é testada, caso o resultado do teste seja verdadeiro, o algoritmo executa os comandos internos ao laço que são caracterizados pela indentação.

Após cada execução do bloco de comandos, o algoritmo testa se a condição é verdadeira, caso positivo, os comandos internos ao laço são executados novamente. Por outro lado, caso a condição testada seja falsa, o algoritmo sai do laço e executa os comandos externos, caracterizados pela ausência da indentação, e finaliza a execução do algoritmo.

Exemplo de programa com laço condicional com teste no início

Enunciado: implemente um algoritmo utilizando laço condicional com teste no início que leia a idade de várias pessoas e no final apresente a média das idades. O critério de parada será a digitação

de uma idade igual a zero ou negativa, em outras palavras, uma idade menor ou igual a zero.

A figura 9.10 apresenta o pseudocódigo (à esquerda) e o código Python (à direita) que implementam a solução do enunciado anterior, cabe ressaltar que esse pseudocódigo já foi explicado em detalhes e, aqui, daremos ênfase ao código Python.

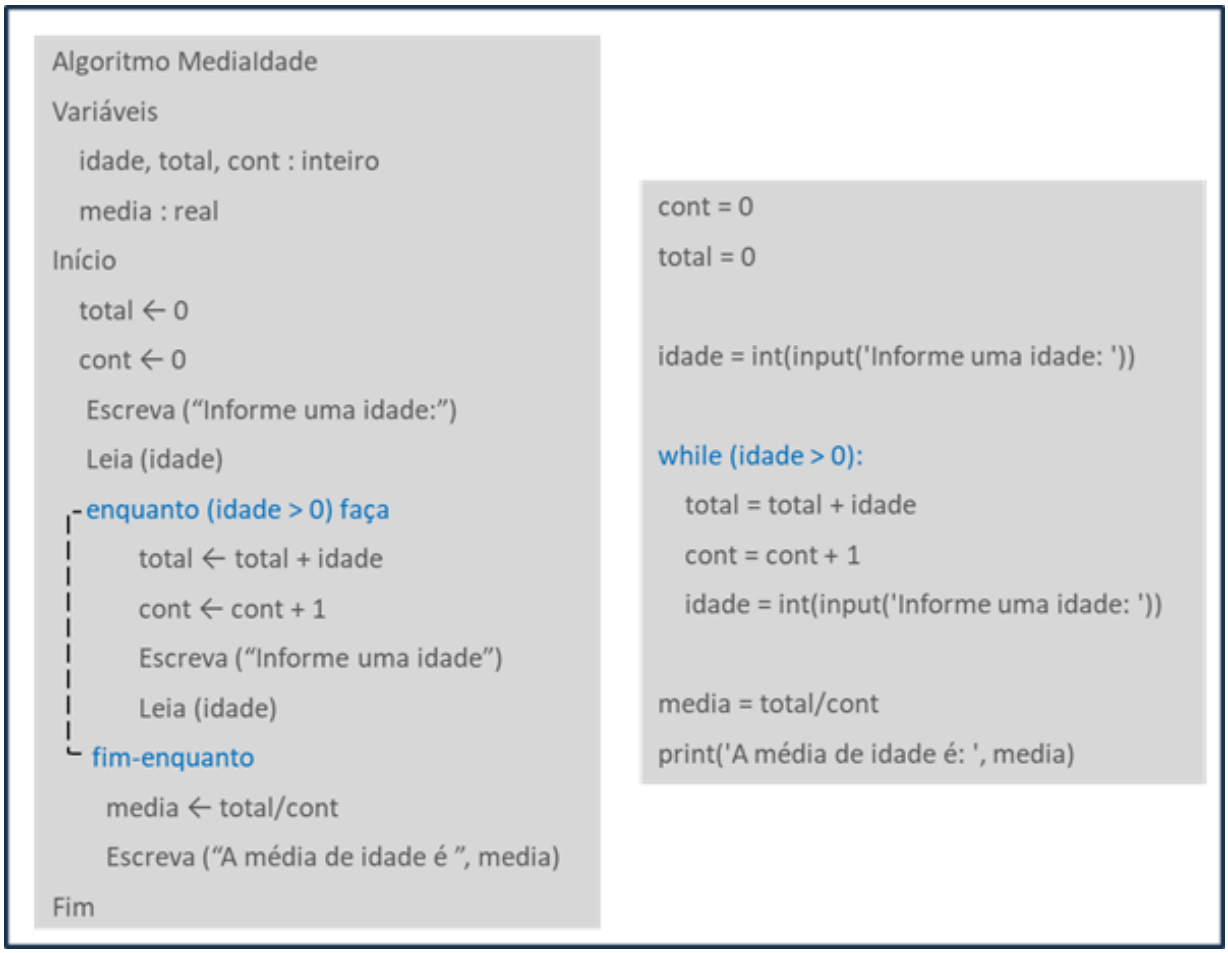


Figura 9.10: Exemplo de algoritmo com laço condicional com teste no início.

```
cont = 0
total = 0

idade = int(input('Informe uma idade: '))

while (idade > 0):
    total = total + idade
```

```
    cont = cont + 1
    idade = int(input('Informe uma idade: '))

media = total/cont
print('A média de idade é: ', media)
```

Como pode ser observado no código em Python da figura 9.10, a variável `idade` armazenará cada idade digitada pelo usuário, a variável `total` armazenará a soma de todas as idades digitadas, a variável `cont` armazenará a quantidade de idades digitadas e a variável `media` armazenará a média das idades digitadas, esse valor será obtido dividindo o valor armazenado na variável `total` pelo valor armazenado na variável `cont`.

A execução do algoritmo é iniciada atribuindo o número zero às variáveis `cont` e `total`. O objetivo é garantir que essas variáveis não contenham valores armazenados na memória do computador utilizados em outros programas.

O próximo passo consiste em solicitar que o usuário digite um valor para ser atribuído à variável `idade`, por meio do comando `idade = int(input('Informe uma idade: '))`. Após a variável `idade` receber o valor digitado pelo usuário, o algoritmo verificará a condição inicial do teste do laço, isto é, se a idade digitada é maior que zero, pois a condição de parada da execução do laço é a idade ser menor ou igual a zero.

Caso essa idade seja maior que zero, o resultado do teste da condição será verdadeiro e o algoritmo executará os comandos internos ao laço, ou seja, os comandos indentados, são eles: `total = total + idade`, que armazena o somatório das idades, `cont = cont + 1`, que armazena a quantidade de idades digitadas, e `idade = int(input('Informe uma idade: '))`, que solicita ao usuário digitar uma nova idade.

Após o usuário digitar a nova idade, o ciclo do laço se repete, quer dizer, o algoritmo testará a idade e, se ela for maior que zero,

executará os comandos internos do laço. Essa repetição só parará quando uma idade menor ou igual a zero for digitada pelo usuário.

Quando o usuário digitar uma idade menor ou igual a zero, a execução do algoritmo sairá do laço, ou melhor, executará os comandos sem indentação, são eles: `media = total/cont` e `print('A média de idade é: ', media)`. O primeiro comando externo ao laço calcula a média das idades e armazena na variável `media`, valor que é calculado dividindo o valor armazenado na variável `total` pelo valor armazenado na variável `cont`. O segundo comando externo apresenta o conteúdo da variável `media` no monitor do computador ao usuário, finalizando a execução do programa.

Após a apresentação dos fundamentos da linguagem de programação Python, finalizamos o aprendizado sobre essa linguagem. Cabe ressaltar que existem vários recursos utilizados para a implementação de algoritmos em Python além dos apresentados neste livro, porém optamos por apresentar apenas os fundamentos da linguagem a fim de que você possa ler, entender e implementar algoritmos básicos escritos em Python.

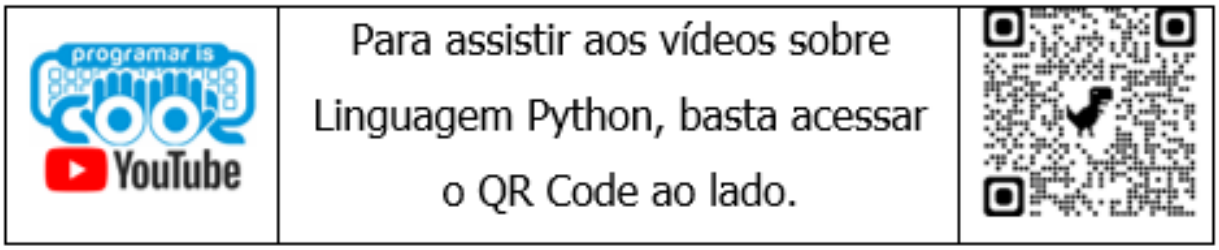


Figura 9.11: Vídeos sobre Linguagem Python. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cICa0OyYNO8YfjXKE1IuJgfJ

Capítulo 10
Ciência de Dados

Confúcio não foi um cientista de dados ou sequer conheceu as técnicas computacionais dessa área. Na verdade, ele foi um famoso pensador que viveu 500 anos antes de Cristo, muito antes da criação do computador. No entanto, ele sintetizou a essência da Ciência de Dados por meio da afirmação "*Se queres prever o futuro, estuda o passado.*".

Ratificando a afirmativa de Confúcio, para usar as técnicas de Ciência de Dados em uma organização com o objetivo de prever (estimar) possíveis cenários futuros com uma determinada probabilidade de acerto é de fundamental importância possuir e entender os dados sobre o seu passado. Assim, as organizações orientadas a dados, além de analisarem o seu passado e presente por meio dos sistemas de apoio a decisão, elas agora se esforçam, também, para prever (estimar) o seu futuro por meio do uso das técnicas de Ciência de Dados.

Essas técnicas surgiram devido à demanda existente no mercado por profissionais capazes de lidar com o cenário *Big Data* e a consequente necessidade de obtenção de *insights* a partir de grandes volumes de dados heterogêneos, gerando vantagens competitivas para as organizações.

Finzer (2013) atribui o surgimento do termo "Ciência de Dados" ao autor William S Cleveland, concebido em 2001 na obra "*Data science an action plan for expanding the technical áreas of the field of statistics*". No entanto, seu início remonta à década de 1960, mais precisamente, em 1962, quando o estatístico norte americano John W. Tukey já defendia a necessidade da realização de análises a partir de dados (PRESS, 2013).

Ciência de Dados consiste em uma área altamente interdisciplinar, embasada em técnicas e teorias advindas de diversos campos das ciências básicas (PORTO; ZIVIANI, 2014). Esse é um campo que combina, principalmente, as habilidades de matemática, estatística, computação e negócio.

Várias ferramentas são utilizadas para a implementação de soluções de Ciência de Dados, dentre as mais comuns, estão as ferramentas de Inteligência Artificial (*Artificial Inteligence*), Aprendizado de Máquina (*Machine Learning*) e Aprendizado Profundo (*Deep Learning*).

10.1 Inteligência Artificial, Aprendizado de Máquina e Aprendizado Profundo

A Inteligência Artificial (IA) surgiu na década de 1950, quando um grupo de pioneiros do campo nascente da Ciência da Computação começou a perguntar se os computadores poderiam ser feitos para "Pensar" — uma pergunta cujas ramificações ainda são exploradas até hoje (CHOLLET, 2018).

Uma definição concisa para o campo seria “o esforço para automatizar tarefas intelectuais normalmente executadas por humanos” (CHOLLET, 2018). IA é um campo geral que engloba, entre outras ferramentas, Machine Learning e Deep Learning, indo além das abordagens que envolvem aprendizado. Os primeiros programas de xadrez, por exemplo, envolviam apenas regras codificadas criadas por programadores e não se qualificaram como aprendizado de máquina.

A figura 10.1 exibe o relacionamento entre a Inteligência Artificial, Machine learning e Deep Learning, de acordo com Chollet (2018).

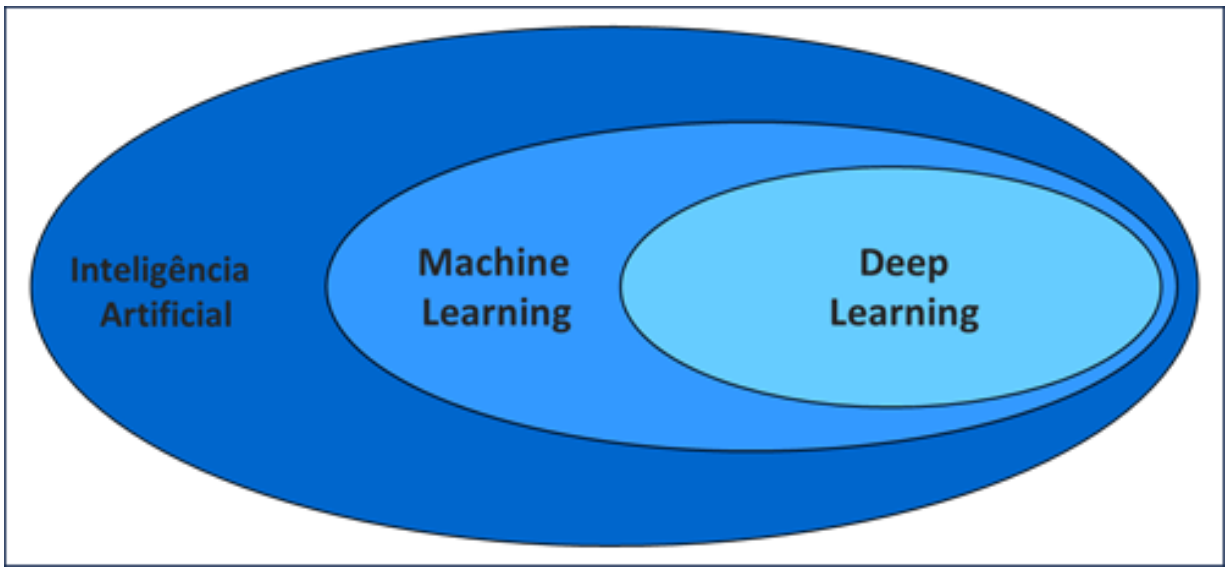


Figura 10.1: Inteligência Artificial, Machine Learning e Deep Learning.

Por um longo tempo, muitos pesquisadores acreditavam que a Inteligência Artificial, em nível humano, poderia ser alcançada por meio da criação de um conjunto suficientemente grande de regras explícitas para manipular o conhecimento. Essa abordagem ficou conhecida como IA simbólica. Era o paradigma dominante na IA desde os anos 50 até o final dos anos 80 e atingiu seu pico de popularidade quando os softwares que tentavam simular o raciocínio de um profissional especialista em alguma área de conhecimento específica começaram a ser desenvolvidos. Embora a IA simbólica tenha se mostrado adequada para resolver problemas lógicos bem definidos, como jogar xadrez, mostrou-se inapropriada para resolver problemas mais complexos e confusos, como classificação de imagem e processamento de linguagem natural, por exemplo.

Então, uma nova abordagem surgiu para complementar a IA simbólica: o Machine Learning (ML). Segundo Chollet (2018), o ML surgiu a partir das indagações:

- Um computador poderia ir além do que é programado para executar e aprender sozinho como realizar uma tarefa específica?
- Um computador poderia surpreender?

- Em vez de programadores elaborando processamento de dados e regras manualmente, um computador poderia aprender automaticamente essas regras observando os dados?

Essas questões abrem as portas para um novo paradigma de programação, pois, enquanto na programação clássica os seres humanos implementam regras em um programa e inserem dados para serem processados de acordo com essas regras para obter respostas, no ML, os seres humanos inserem dados, bem como as respostas esperadas, e o resultado do processamento são as regras (em forma de modelos). Essas regras podem ser aplicadas a novos dados para produzir resultados originais. A figura 10.2 apresenta esses dois paradigmas de programação.

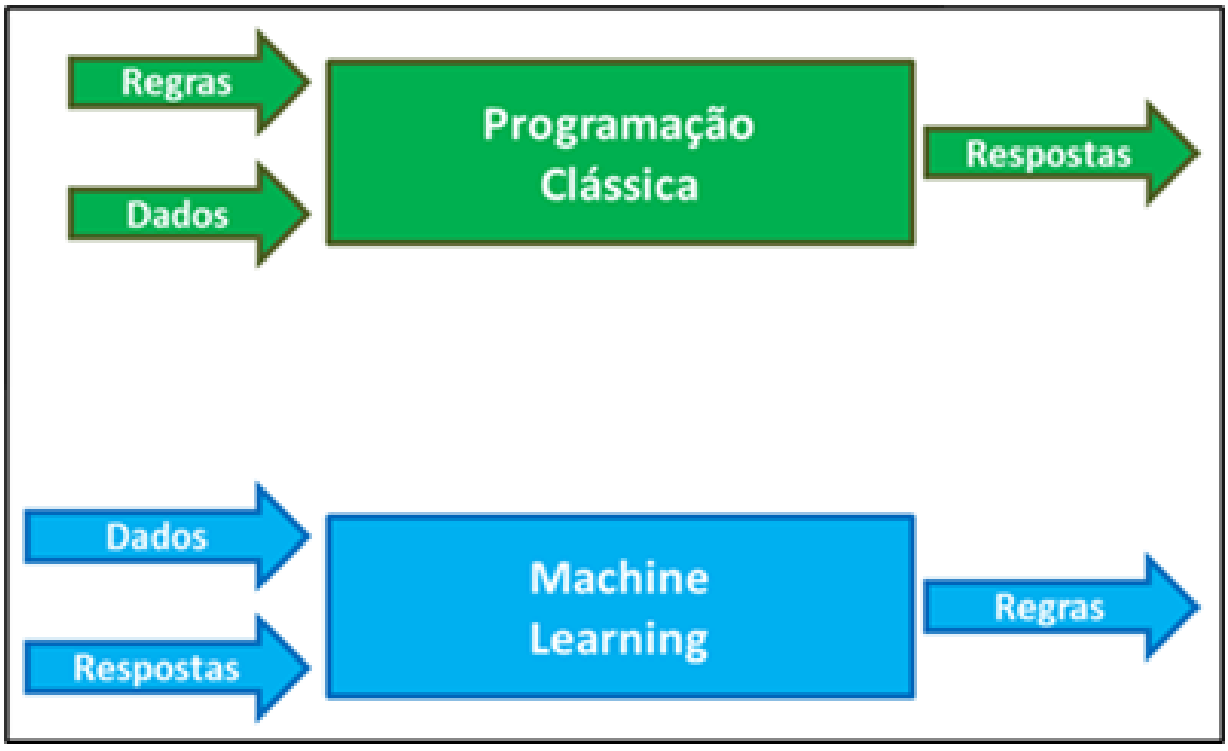


Figura 10.2: Programação clássica e para Machine Learning.

Programação clássica *x* Machine Learning por meio de um exemplo

Para tornar os dois paradigmas de programação abordados na figura 10.2 mais claros, vamos exemplificar o uso de cada um desses paradigmas em uma empresa administradora de cartão de crédito. A figura 10.3 apresenta um exemplo de sistema transacional, que utiliza a programação clássica para a implementação de um programa para emitir a fatura mensal do cartão de crédito dos clientes.

Figura 10.3: Programação clássica.

Como pode ser observado na figura 10.3, primeiramente foi desenvolvido um programa transacional que implementa as regras de negócio para gerar a fatura mensal dos clientes de uma administradora de cartão de crédito. Essas regras, de maneira simplificada, são: somar o valor da mensalidade do cartão, somar o valor das compras realizadas pelo cliente no mês corrente, somar saldo não pago da fatura do mês anterior e aplicar os juros nas situações previstas.

Com o programa desenvolvido, todo mês a administradora do cartão inserirá para cada cliente a mensalidade do cartão; as compras realizadas no mês; saldo não pago da fatura anterior; e os juros, por sua vez, o programa fará os cálculos e gerará a fatura mensal de cada cliente como resultado. Esse é um típico exemplo da programação clássica.

Considerando a mesma administradora de cartão de crédito, o primeiro passo para desenvolvermos uma aplicação de ML consiste em escolher o problema de negócio a ser abordado. Neste caso, simularemos o processo de análise para conceder ou não um cartão de crédito a um novo cliente.

Inicialmente, são estabelecidas duas classes de clientes, o bom pagador, cliente que paga pontualmente as faturas mensais, e o mau pagador, cliente que atrasa ou não paga as faturas mensais. O critério para conceder um cartão a um novo cliente é fazer a predição (estimar) para saber se ele tende a ser um bom pagador. Essa predição pode ser feita por meio da técnica de classificação utilizando uma árvore de decisão, embora existam outras técnicas e outros tipos de modelos.

A árvore de decisão é um tipo de modelo que estabelece nós organizados em uma hierarquia que são avaliados para se chegar a um resultado. Duas categorias de nós se destacam, o nó-raiz, que é o ponto de partida para a análise, e os nós-folha, que são os possíveis resultados. No contexto do processo escolhido, o nó-folha é a classe que será gerada como resposta e os demais nós representam os atributos da base de dados utilizados para a análise.

No processo escolhido, o resultado é o comportamento de um cliente que pode ser bom ou mau pagador. A árvore é gerada utilizando os dados existentes na empresa, compostos pelos registros de milhares ou milhões de clientes e respectivos pagamentos das suas faturas ao longo de décadas.

Então, quando um novo cliente solicitar um cartão de crédito, seus atributos serão comparados com os nós da árvore de decisão gerada para verificar a possibilidade de ele ser um bom ou mau pagador. A partir dessa comparação, a empresa decidirá se concederá ou não um cartão ao novo cliente.

Para a construção da árvore de decisão descrita anteriormente, foram utilizados dados sobre cadastro de clientes e pagamento de fatura. A figura 10.4 exibe uma parte do conjunto de dados sobre os clientes atuais da empresa.

Nome	Idade	Curso Superior	Renda Mensal	Bom Pagador
Ana	25	Sim	Alta	True
Beto	28	Não	Baixa	False
Caio	35	Não	Alta	True
Edu	42	Sim	Baixa	False
...	...	...	...	...

Figura 10.4: Dados sobre clientes.

O conjunto de dados a ser utilizado possui as seguintes informações sobre os clientes atuais da administradora de cartão de crédito: `Nome`, `Idade`, `Curso Superior`, `Renda Mensal` e `Bom Pagador`, que significam, respectivamente, o nome do cliente; sua idade; se possui curso superior; a sua renda mensal, que pode ser `Alta`, `Média` ou `Baixa`; e se o cliente é bom pagador, ou seja, se mantém o pagamento de suas faturas em dia. A figura 10.4 apresenta apenas as quatro primeiras linhas desse conjunto de dados formado por centenas de milhares de linhas.

Cabe ressaltar que esses dados são obtidos a partir dos sistemas transacionais da empresa, pois trata-se de uma junção de dados

cadastrais dos clientes e dados financeiros relativos a pagamentos das faturas mensais.

A figura 10.5 apresenta a implementação de uma aplicação de ML que utiliza o conjunto de dados exibido na figura 10.4 para gerar um modelo, neste caso uma árvore de decisão, para futuramente comparar os dados de um pretendente a possuir um cartão de crédito e prever (estimar) se esse potencial cliente tende a ser um bom ou mau pagador.

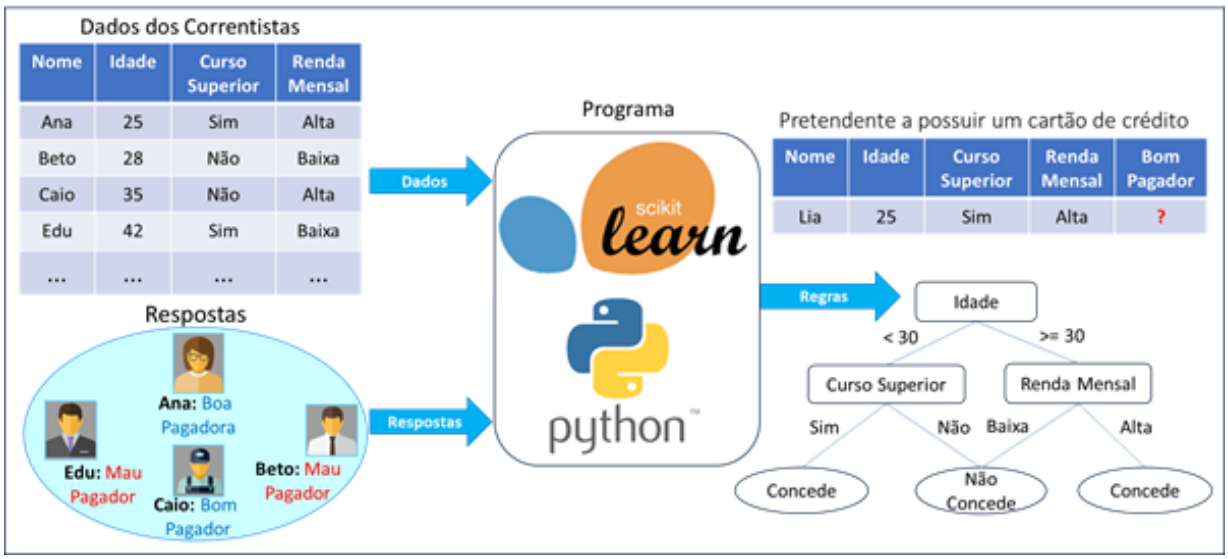


Figura 10.5: Programação para Machine Learning.

À esquerda e acima da figura 10.5, estão os dados sobre os correntistas e, logo abaixo, a indicação se são bons ou maus pagadores. Esses dados são processados pelo programa *scikit-learn*, que gera a árvore de decisão localizada à direita da mesma figura.

Diferentemente da programação clássica apresentada na figura 10.3, uma aplicação de ML não foca a implementação de regras de negócio por meio de um programa, que será utilizado para gerar um resultado. Em ML, o resultado consiste em um modelo, no caso da figura 10.5 o resultado foi a geração de uma árvore de decisão.

Como pode ser observado na figura 10.5, não há a necessidade de implementação de um programa para aplicações de ML, pois as

linguagens de programação já disponibilizam pacotes ou bibliotecas prontas para esse tipo de aplicação. Em Python, por exemplo, existe a biblioteca *scikit-learn* que foi desenvolvida especificamente para aplicação prática de ML.

Sendo assim, de acordo com a figura 10.5, a pessoa cientista de dados prepara um conjunto de dados composto por atributos de vários clientes e as respectivas respostas, ou seja, se o cliente é um bom ou mau pagador, submete esses dados a um pacote existente no *scikit-learn*. Esse programa, por sua vez, gera uma árvore de decisão baseado nos dados e respostas fornecidos. A árvore de decisão gerada permite a predição (estimativa) do comportamento de novos clientes a partir de seus atributos.

Então, quando um potencial cliente solicitar um cartão de crédito, a administradora solicitará os seus dados e percorrerá a árvore de decisão com esses dados para verificar se ele tende a ser um bom ou mau pagador. Caso a estimativa seja que o cliente tende a ser um bom pagador, a administradora fornecerá um cartão, caso contrário, se o cliente for estimado como um mau pagador, a administradora não fornecerá o cartão. Cabe ressaltar que ML trabalha com estimativa, não é certeza.

Como um exemplo de predição, à direita e acima da figura 10.5, a pretendente `Lia` submete uma solicitação de cartão de crédito à administradora e fornece seus dados para análise. Comparando os seus atributos com os nós da árvore de decisão gerada, será estabelecido o seguinte percurso: a `Idade` será o primeiro nó a ser percorrido, como a `Lia` tem menos de 30 anos, o percurso irá para o nó à esquerda, que representa `Curso Superior`, como a `Lia` possui curso superior, o resultado será `Conceder` o cartão de crédito, pois ela, segundo à árvore de decisão, tende a ser uma boa pagadora.

Deep Learming

Além de Machine Learning, a IA abrange as técnicas de Deep Learming (DL). Segundo Chollet (2018), DL é um subcampo específico do ML, é uma nova visão das representações do aprendizado a partir de dados que enfatizam o aprendizado de camadas sucessivas de representações significativas.

Em DL, o número de camadas que contribuem para um modelo de dados é chamado de profundidade do modelo. O aprendizado geralmente envolve várias camadas sucessivas de representações. Outras abordagens de ML tendem a se concentrar em usar apenas uma ou duas camadas de representações dos dados; portanto, às vezes são chamados de Aprendizado Superficial (*Shallow Learning*) (CHOLLET, 2018).

As representações em camadas são (quase sempre) aprendidas via modelos chamados redes neurais artificiais, estruturados em camadas empilhadas umas sobre as outras. O termo "rede neural" é uma referência à neurobiologia.

Chollet (2018) afirma que embora alguns dos conceitos em DL tenham sido desenvolvidos em parte inspirando-se no entendimento do cérebro, os modelos de DL não são modelos do cérebro. O autor assegura que não há evidências de que o cérebro implemente algo como os mecanismos de aprendizado usado em modelos modernos de DL. Na verdade, DL é uma estrutura matemática para aprender representações a partir de dados.

A figura 10.6 apresenta um exemplo de uso de DL para o reconhecimento de um dígito numérico a partir de uma imagem.

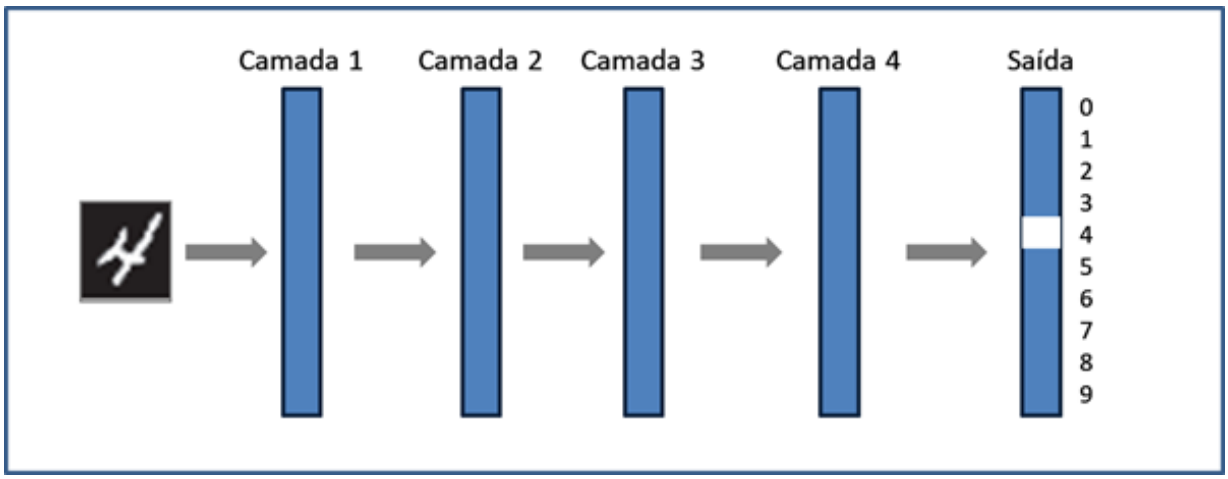


Figura 10.6: Uso de DL para o reconhecimento de um dígito numérico.

Conforme mostrado na figura 10.6, uma imagem é apresentada ao modelo de DL para ser identificada como um dígito entre 0 e 9. Antes da análise da imagem, o modelo passou por um massivo processo de treinamento, que consiste em apresentar ao modelo várias maneiras diferentes de representar esses dígitos. Para a identificação, a imagem passa por várias camadas de processamento com algoritmos e, após esse processamento, o modelo considera o número 4 como o dígito mais parecido com a imagem. Como se nota, a figura não contém exatamente o número 4, mas devido ao fato de ML trabalhar na maior parte das vezes com estimativa, existe uma grande possibilidade de a imagem representar o número 4, pois ela está mais parecida com esse dígito, se comparada aos demais. Assim sendo, o modelo de DL considera que a imagem é o dígito 4, embora não seja exatamente esse dígito.

Após esta breve introdução sobre Ciência de Dados e seu relacionamento com a IA, torna-se importante entender os principais tipos de aprendizagem em ML para a implementação de projetos de Ciência de Dados.

10.2 Tipos de aprendizado

Os principais tipos ou formas de aprendizado em ML empregados em projetos de Ciência de Dados são:

- Aprendizado supervisionado;
- Aprendizado não supervisionado;
- Aprendizado por reforço;

O **aprendizado supervisionado** é caracterizado pelo uso de conjuntos de dados para treinar algoritmos que categorizam coisas (classificação) ou estimam valores (regressão). Um exemplo típico de aprendizado supervisionado é a classificação de clientes de uma administradora de cartão de crédito apresentada na figura 10.5.

Nesse exemplo, inicialmente, foram estabelecidas duas classes de clientes, os bons pagadores e os maus pagadores, embasado nos dados que a administradora detém sobre os clientes que já possuem cartão de crédito. Após esse passo, foi criado um conjunto de dados composto por vários atributos desses clientes e a classe a que cada um deles pertence, pois a empresa possui essas informações em seus sistemas transacionais. Em seguida, o conjunto de dados criado foi utilizado para o treinamento do algoritmo *scikit-learn*. Após o treinamento, o *scikit-learn* gerou uma árvore de decisão como resultado. Essa árvore de decisão pode ser utilizada para fazer a análise de novos clientes a partir da comparação de seus atributos com os nós de decisão da árvore para verificar a possibilidade de conceção ou não de um cartão de crédito, proporcionando a predição.

Por outro lado, o **aprendizado não supervisionado** ocorre quando um algoritmo tenta encontrar padrões e relações em um conjunto de dados, sem um treinamento prévio. O objetivo do algoritmo de aprendizado não supervisionado é categorizar os dados de alguma forma ou descrever sua estrutura. Nesse tipo de aprendizado, o algoritmo explora dados para descobrir padrões e estruturas ocultas. Uma técnica comum para isso é o agrupamento (clusterização), que é o processo de agrupar ou categorizar grupos de dados.

Considerando o mesmo conjunto de dados da figura 10.5, podemos submetê-lo a um algoritmo de agrupamento para esse algoritmo sugerir grupos de clientes da administradora de cartão de crédito por meio da identificação de padrões e semelhança entre os clientes. O objetivo de aplicar o agrupamento é interpretar os grupos de clientes sugeridos pelo algoritmo. Após a interpretação, é possível concluir, por exemplo, que existem grupos de clientes que tendem a comprar o mesmo tipo de produto ou que gastam mensalmente valores semelhantes.

Assim, a grande diferença entre aprendizado supervisionado e o não supervisionado é o treinamento do modelo por meio de um conjunto de dados. Na classificação, que é uma técnica de aprendizado supervisionado, o cientista de dados estabelece as classes e treina o algoritmo utilizando um conjunto de dados. O algoritmo gera como resultado um modelo que poderá classificar novos dados em uma das classes estabelecidas. No agrupamento, que é uma técnica de aprendizado não supervisionado, o conjunto de dados é apresentado ao algoritmo para o algoritmo sugerir os grupos, depois disso, o cientista de dados tenta interpretar as características de cada grupo.

Finalmente, o **aprendizado por reforço** é caracterizado pelo treinamento de modelos de ML para tomar decisões. No aprendizado por reforço, um agente de software é treinado para tomar sequências de decisões, recebendo recompensas ou sendo penalizado como resultado. O objetivo é que esse agente aprenda a executar as melhores ações para maximizar as recompensas ao longo do tempo. Esse tipo de aprendizado é amplamente aplicado em robótica, carros autônomos e jogos.

Um exemplo da utilização do aprendizado por reforço é o treinamento do software controlador de carros autônomos. Diferentes vias, regras de trânsito, atitudes dos pedestres e diversos elementos tornam o ambiente bastante complexo e desafiador. As equipes de IA, inicialmente, treinam o software em um ambiente

simulado. O algoritmo realiza ações com base no ambiente apresentado e recebe recompensas ou multas.

Com o passar do tempo, após percorrer milhões de quilômetros em diversos ambientes virtuais, o carro autônomo adquire conhecimento sobre as melhores ações a serem tomadas em cada situação, sem precisar simular toda a complexidade do tráfego. Ao ser colocado na prática, o carro utiliza as estratégias aprendidas, porém continua a aprimorá-las com informações atualizadas. Assim, o aprendizado evolui constantemente.

Em projetos de Ciência de Dados, além dos tipos, existem as técnicas de aprendizagem em ML.

10.3 Técnicas utilizadas em Machine Learning

As técnicas utilizadas em ML podem ser categorizadas em dois grupos: preditivas e descritivas. As técnicas **preditivas** são voltadas para previsão (estimativa) por meio da busca por uma função ou modelo capaz de generalizar algo. Por outro lado, as técnicas **descritivas** exploram, associam ou descrevem um conjunto de dados. A figura 10.7 apresenta as principais técnicas de ML organizadas nessas duas categorias.

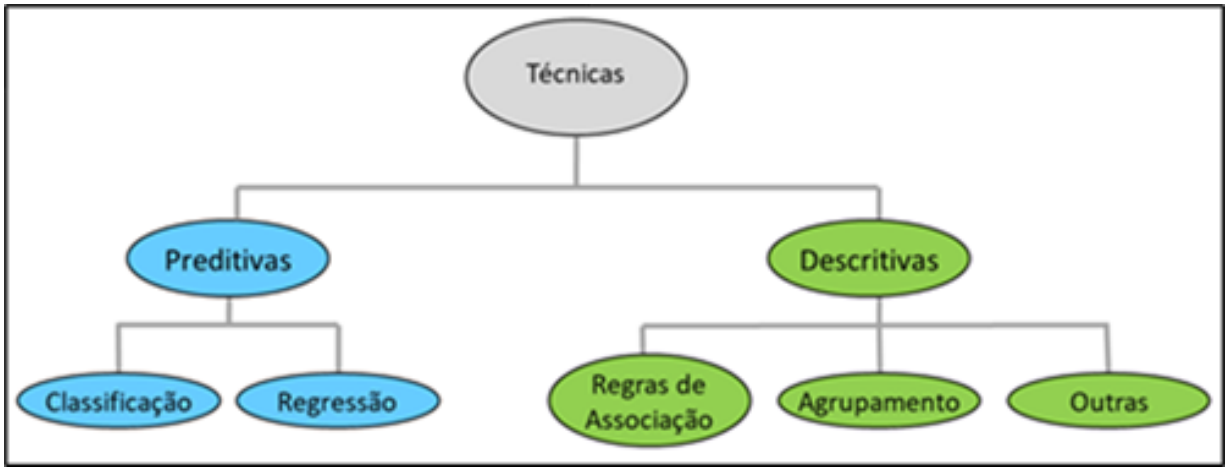


Figura 10.7: Técnicas utilizadas em Machine Learning.

Como pode ser observado na figura 10.7, as técnicas de ML preditivas são classificação e regressão e as técnicas descritivas mais usuais são regras de associação e agrupamento, embora existam outras técnicas descritivas.

A classificação, como já apresentado anteriormente, tem por objetivo categorizar registros com o intuito de prever, por exemplo, se um indivíduo vai ou não pertencer a uma determinada classe de consumidor com base nas suas características pessoais.

Por outro lado, a regressão tem a finalidade de prever números e não classes. As empresas administradoras de cartão de crédito utilizam essa técnica para aumentar ou reduzir o limite de crédito de seus clientes.

Diferentemente da classificação e da regressão, que possuem o propósito de prever classes ou valores, as regras de associação e o agrupamento buscam encontrar padrões.

As regras de associação visam descobrir padrões associativos entre registros pertencentes a um conjunto de dados. Podem, por exemplo, ser utilizadas para identificar produtos ou serviços que sempre são vendidos na mesma transação. Assim, produtos ou

serviços complementares são sugeridos durante uma compra para melhorar a experiência do cliente.

O agrupamento, já apresentado neste capítulo, tem como meta analisar todos os registros de um conjunto de dados e agrupar aqueles que possuem características semelhantes, ou seja, que pertençam ao mesmo grupo.

Embora existam outras técnicas utilizadas em Machine Learning, as técnicas descritas anteriormente são suficientes para proporcionar o entendimento dos fundamentos da Ciência de Dados, que é o objetivo deste livro. No entanto, durante os cursos que ministro sobre esse tema, vários alunos solicitam maiores detalhes sobre os assuntos abordados. Por isso, decidi apresentar nos próximos capítulos alguns detalhes dessas técnicas que vão além da alfabetização de dados, como fórmulas e funções matemáticas. Então, o leitor ou leitora que não tiver interesse em assimilar essas particularidades pode focar apenas no texto dos próximos capítulos e pular o conteúdo mais técnico.

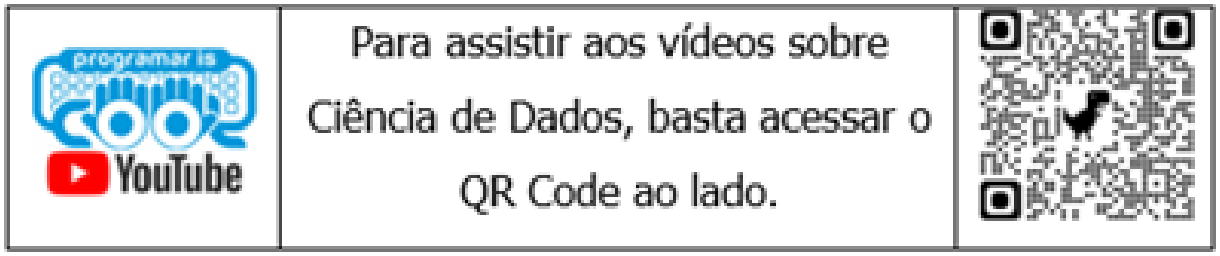


Figura 10.8: Vídeos sobre Ciência de Dados. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIDgRIMMSTEXGOyfS2MlTHNE

Referências

CHOLLET, Francois. *Deep learning with Python*. Simon and Schuster, 2021.

FINZER, William. The data science education

dilemma. *Technology Innovations in Statistics Education*, California, v. 7, n. 2, p. 1-9, 2013.

PORTO, Fábio; ZIVIANI, Artur. Ciência de dados. *In: SEMINÁRIO DE GRANDES DESAFIOS DA COMPUTAÇÃO NO BRASIL*, 3., 2014, Rio de Janeiro. Anais. Rio de Janeiro: [SBC], 2014. p. [1-18].

PRESS, Gil. A very short story of data science. *Forbes*, [S.l.], May 28, 2013.

Capítulo 11
Aprendizado supervisionado

Neste capítulo serão apresentadas as duas técnicas de aprendizado supervisionado: a classificação e a análise de regressão. Cabe ressaltar que não há a necessidade de dominar as funções e fórmulas matemáticas apresentadas para ser considerada uma pessoa alfabetizada em dados, mas sugiro a você, pelo menos, que leia esse conteúdo e assimile o que for possível, pois aprender, além de útil, nunca é demais.

11.1 Classificação

Classificação é o processo de aprendizado supervisionado que envolve categorizar ou rotular dados com base em suas características. Nessa técnica, um algoritmo é treinado usando um conjunto de dados rotulado, ou seja, com as classes atribuídas a cada item de dado, para gerar um modelo capaz de mapear (classificar) um novo item de dado para uma das várias classes predefinidas. O objetivo desse modelo é classificar corretamente novos itens de dados com base em suas características, possibilitando a predição.

São exemplos de aplicações de classificação em Ciência de Dados:

- Classificação de spam em e-mails.
- Diagnóstico médico com base em sintomas.
- Previsão de evasão de clientes ou funcionários em empresas.
- Reconhecimento de padrões em imagens para identificar objetos.

- Classificação de transações financeiras como legítimas ou fraudulentas.

Os principais métodos utilizados em classificação são:

- Métodos baseados em distância.
- Métodos probabilísticos.
- Métodos baseados em procura.
- Métodos baseados em otimização.
- Métodos ensemble (mistos).

Métodos baseados em distância

Os métodos de classificação baseados em distância, consideram a proximidade entre os itens de um conjunto de dados para a realização de predições. O algoritmo *K Nearest Neighbors* (K-NN), que significa K-vizinhos mais próximos, parte da hipótese de que itens de dados similares tendem a estar concentrados na mesma região no espaço de dispersão dos dados.

Nesse algoritmo, o K representa o número de vizinhos a serem considerados para a classificação de um novo item de dado e a distância pode ser baseada em várias métricas, dentre elas a distância euclidiana. Cabe ressaltar que assim como o valor de K, a medida de distância influencia diretamente a performance de modelos criados com o algoritmo K-NN. A figura 11.1 apresenta a fórmula para o cálculo da distância euclidiana para um ponto no espaço bidimensional.

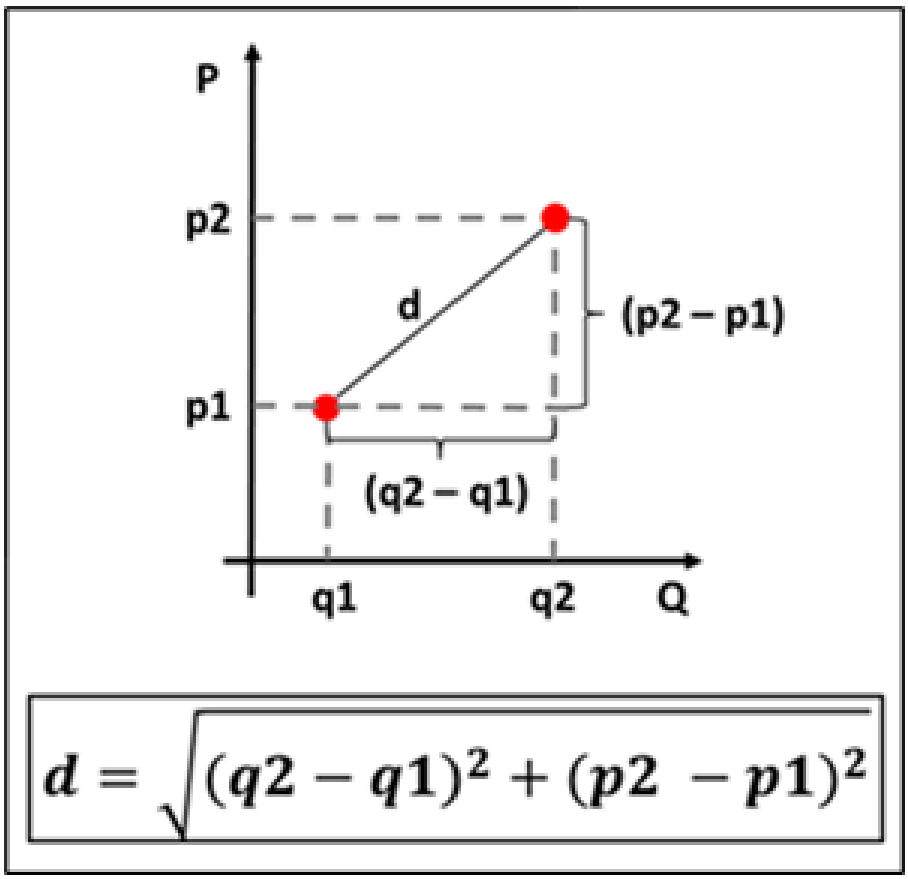


Figura 11.1: Distância Euclidiana.

Como pode ser observado na figura 11.1, a distância euclidiana é baseada no teorema de Pitágoras que afirma que em um triângulo retângulo, o quadrado da hipotenusa é igual à soma dos quadrados dos catetos. No entanto, a distância euclidiana estende esse conceito para espaços com mais de duas dimensões.

Após a obtenção de um conjunto de dados, os passos do algoritmo K-NN são:

- Definir a métrica para cálculo da distância;
- Definir o valor de K;
- Calcular a distância do novo item aos itens do conjunto de dados.
- Contar o número de vizinhos de cada classe mais próximos do item a ser classificado, considerando K como a quantidade total

de vizinhos.
- Classificar o novo item na classe com mais itens próximos.

A figura 11.2 apresenta a representação gráfica em um plano bidimensional de um conjunto de dados que contém duas classes distintas, a classe A, cujos itens de dados são representados por quadrados, e a classe B, cujos itens de dados são representados pelos triângulos.

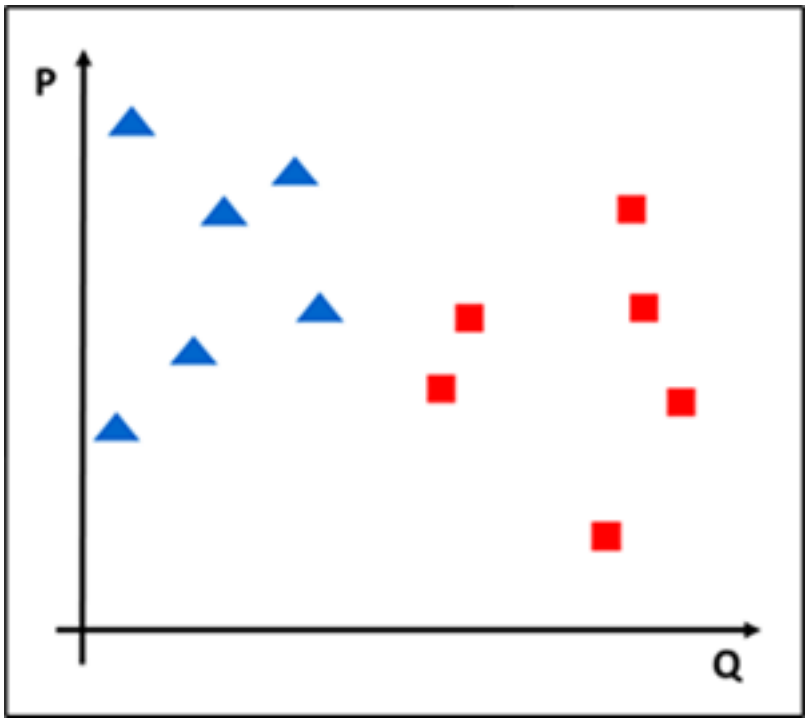


Figura 11.2: Representação gráfica de um conjunto de dados com duas classes.

A figura 11.3 apresenta a utilização do algoritmo K-NN para classificar um novo ponto usando o mesmo conjunto de dados da figura 11.2

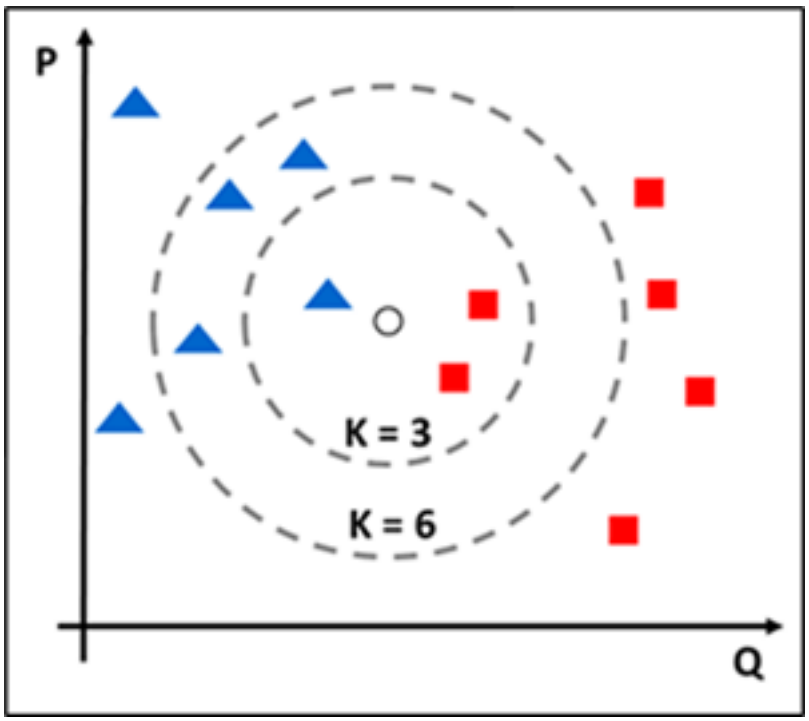


Figura 11.3: Classificação utilizando o algoritmo K-NN.

Como pode ser observado na figura 11.3, considerando a distância euclidiana entre os pontos, quando o $K = 3$, o novo ponto, representado pelo círculo, pertencerá à classe A, pois existem dois quadrados mais próximos e apenas um triângulo. Por outro lado, quando $K = 6$, o novo ponto pertencerá à classe B pois existem quatro triângulos mais próximos e apenas dois quadrados.

Assim, o cientista de dados deve testar as várias opções de K e fazer uma análise para determinar qual valor de K é o mais apropriado para a classificação de um novo item a partir do conjunto de dados utilizado. Outro aspecto importante que deve ser observado durante o uso do algoritmo K-NN é que os dados devem estar padronizados, ou seja, todas as variáveis precisam estar na mesma ordem de grandeza.

Métodos probabilísticos

Um dos métodos probabilísticos mais usuais para a classificação é fundamentado no Teorema de Bayes, denominado classificador

Naïve Bayes. O Teorema de Bayes é um modelo matemática utilizado para calcular a probabilidade de um evento ocorrer dado que outro evento já ocorreu, o que é denominado de probabilidade condicional. A figura 11.4 apresenta esse teorema.

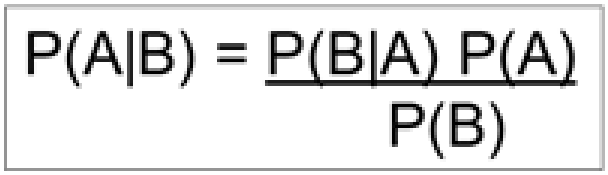


Figura 11.4: Teorema de Bayes.

O Teorema de Bayes apresentado na figura 11.4 pode ser interpretado da seguinte forma:

- `A` e `B` são eventos observados.
- `P(A|B)` : representa a probabilidade de ocorrer o evento `A` dado que o evento `B` ocorreu.
- `P(B|A)` : representa a probabilidade de ocorrer o evento `B` dado que o evento `A` ocorreu.
- `P(A)` : representa a probabilidade de ocorrer o evento `A` .
- `P(B)` : representa a probabilidade de ocorrer o evento `B` .

O classificador Naïve Bayes é um algoritmo utilizado para fazer predições em aprendizagem de máquina. A expressão "naïve" (ingênua) se refere à maneira como o algoritmo examina os atributos de uma base de dados: o algoritmo considera que os atributos são independentes entre si. Esse classificador pode ser utilizado para detectar se alguém tem uma doença ou não, para detectar se um e-mail é um SPAM, em sistemas de recomendação, em sistemas de análise de sentimentos, entre outras aplicações.

A figura 11.5 apresenta um conjunto de dados sobre corridas para exemplificarmos o uso do classificador Naïve Bayes.

Clima	Correr
Ensolarado	Sim
Nublado	Não
Chuvoso	Sim
Ensolarado	Não
Ensolarado	Sim
Nublado	Sim
Chuvoso	Não
Chuvoso	Não
Ensolarado	Sim
Chuvoso	Não
Ensolarado	Não
Nublado	Sim

Figura 11.5: Conjunto de dados sobre corridas.

O conjunto de dados da figura 11.5 é composto por 12 observações com dois atributos. O atributo `Clima` pode possuir os valores `Ensolarado`, `Nublado` ou `Chuvoso` e o atributo `Correr` que pode possuir os valores `Sim` ou `Não`. Esse conjunto de dados será utilizado para treinar um modelo utilizando Naïve Bayes para possibilitar a predição sobre a ocorrência ou não de uma corrida a partir de uma determinada condição climática. O primeiro passo consiste em criar a tabela de frequência relativa ao conjunto de dados disponibilizado, conforme apresentado na figura 11.6.

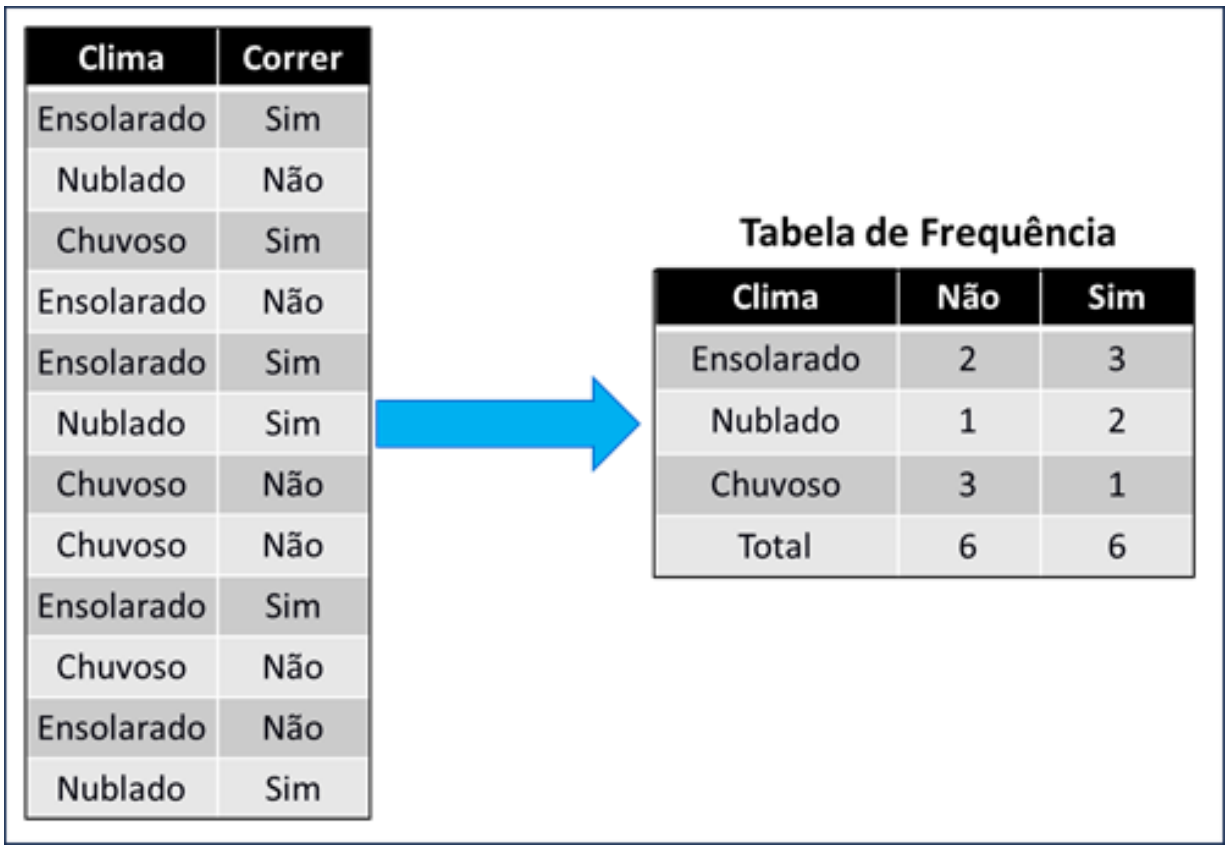


Figura 11.6: Tabela de frequência para o conjunto de dados sobre corridas.

Como pode ser observado na tabela de frequência da figura 11.6, das 12 observações do conjunto de dados em 6 houve corrida e nas outras 6 não houve. Com o clima `Ensolarado` houve corrida 3 vezes e não houve em 2 vezes. Por outro lado, com o clima `Nublado` houve corrida 2 vezes e não houve em 1 vez. Finalmente, com o clima `Chuvoso` houve corrida 1 vez e não houve em 3 vezes.

Após a criação da tabela de frequência, são calculadas as probabilidades de cada ocorrência e de cada combinação, conforme apresentado na figura 11.7.

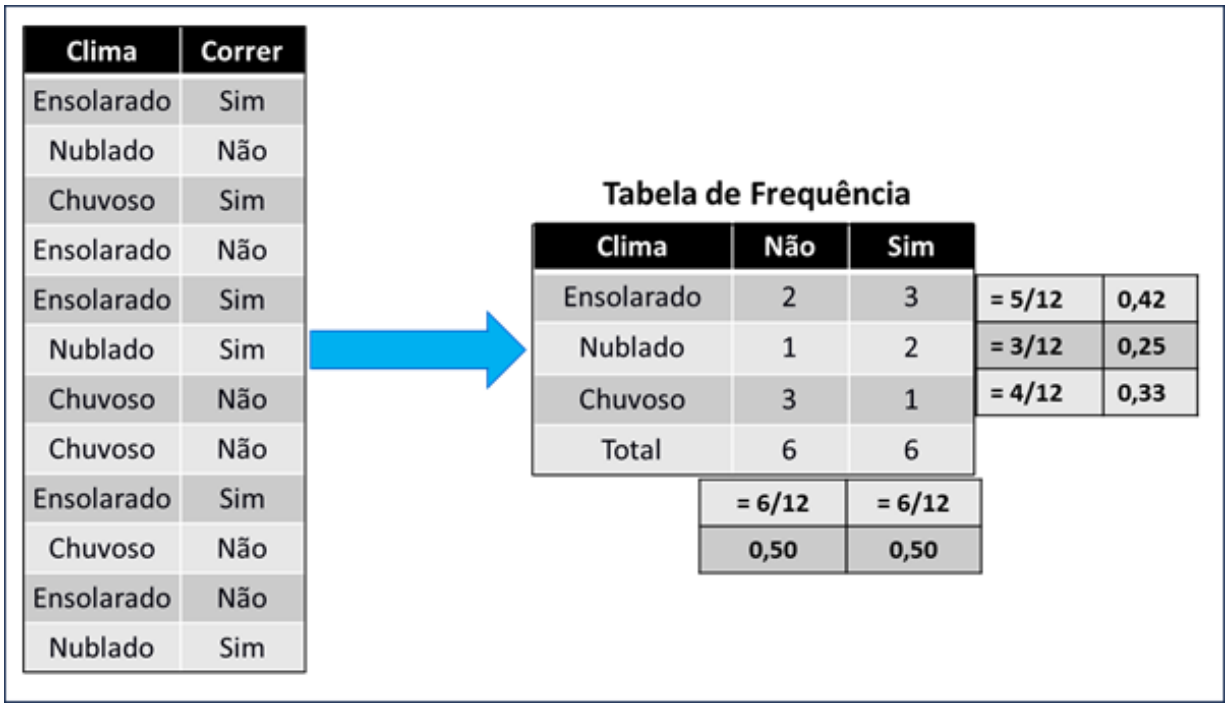


Figura 11.7: Probabilidade das ocorrências para o conjunto de dados sobre corridas.

Como pode ser observado no lado direito da tabela de frequência da figura 11.7, das 12 observações do conjunto de dados 5 apresentam o clima `Ensolarado` (5/12 = 42%), 3 o clima `Nublado` (3/12 = 25%) e 4 o clima `Chuvoso` (4/12 = 33%).

Ainda nesse conjunto de dados, em 50% das observações houve corrida e nos outros 50% não houve, como pode ser observado na parte inferior da mesma tabela de frequência.

Considerando o conjunto de dados apresentado e a tabela de frequência com as respectivas probabilidades de ocorrência de corridas, deseja-se prever se haverá corrida caso o clima esteja ensolarado.

A figura 11.8 apresenta o uso do Teorema de Bayes para fazer a predição solicitada.

Figura 11.8: Uso do Teorema de Bayes para fazer predição.

Os termos apresentados na fórmula da figura 11.8 são:

- `P(Sim|Sol)` : probabilidade de haver corrida, dado que o clima está `Ensolarado` .
- `P(Sol|Sim)` : probabilidade de o clima estar `Ensolarado` , dado que houve corrida = 0,5 (50%), pois sabe-se que houve 6 corridas, dessas 6 corridas, 3 aconteceram em dias ensolarados.
- `P(Sim)` : probabilidade haver corrida = 0,5 (50%), pois sabe-se que houve 6 corridas em 12 observações.
- `P(Sol)` : probabilidade de o clima estar `Ensolarado` = 0,42 (42%), pois sabe-se que houve 12 observações, dessas 12 observações, 5 aconteceram em dias ensolarados.

Assim teremos:

- `P(Sim|Sol) = (0,5 * 0,5) / 0,42 = 0,595 = 59,5 %` .
- Então, caso o atributo clima possua o valor `Ensolarado` haverá a probabilidade de 59.5% de haver corrida.

A partir do exemplo apresentado, pode-se observar que a predição não consiste em um fato, na verdade, trata-se de uma estimativa. Essa estimativa oferece subsídios ao decisor fazendo com que ele evite tomar decisões baseadas apenas no instinto e passe a analisar *insights* proporcionados pela aplicação de técnicas de Ciência de Dados usando como insumo os dados históricos da organização para dar suporte à tomada de decisão.

Métodos baseados em procura

Árvore de decisão é o método baseado em procura mais usual em aprendizado de máquina supervisionado, pode ser utilizada em tarefas de classificação ou regressão, nesta seção será abordado apenas o seu uso para a classificação.

Uma árvore de decisão é composta por elementos chamados *nós*, dentre os quais um é denominado nó raiz. Esse nó origina uma relação de "paternidade" que gera uma estrutura hierárquica sobre os nós. Todos os nós que não possuem filhos são chamados nós terminais ou folhas. Os nós que contêm filhos são chamados nós internos ou de decisão. A figura 11.9 apresenta a estrutura de uma árvore de decisão.

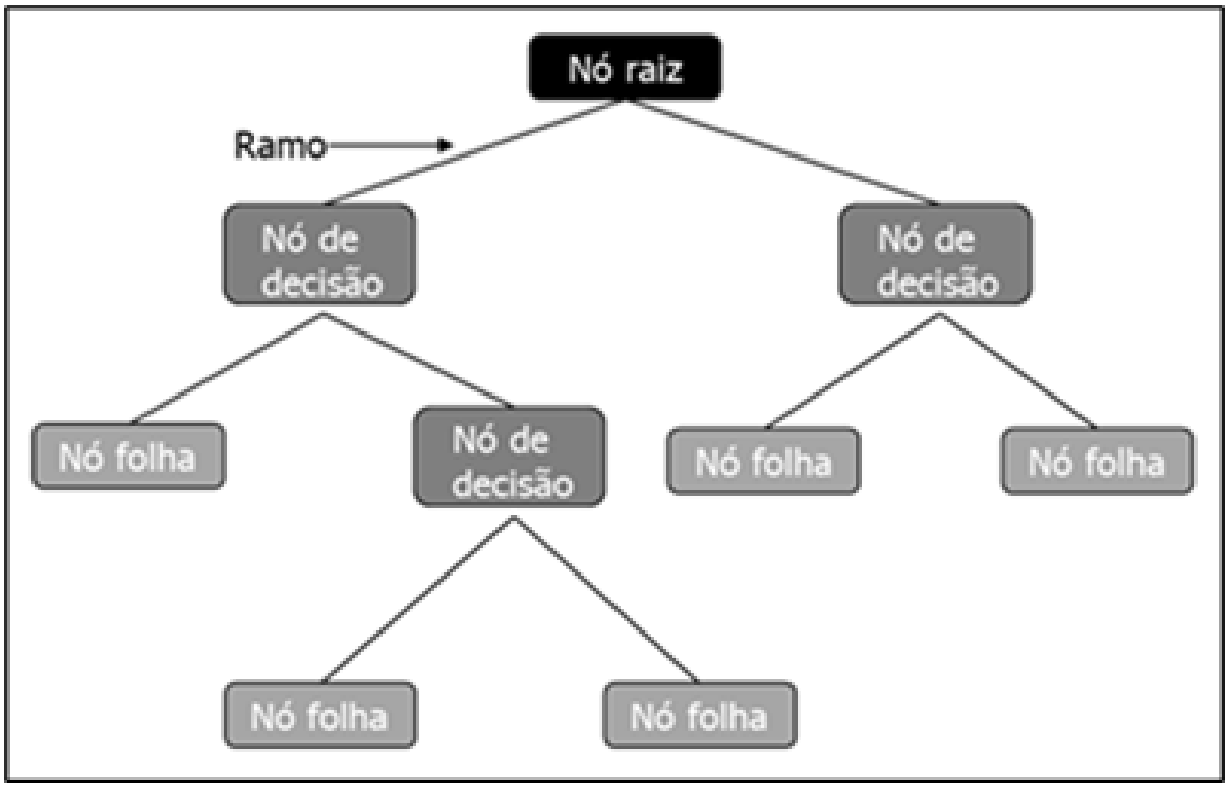


Figura 11.9: Estrutura de uma árvore de decisão.

Como pode ser observado na figura 11.9, uma árvore de decisão possui a seguinte estrutura típica:

- Nó raiz;
- Nós internos ou de decisão: são rotulados com atributos;

- Nós folhas: são rotuladas com classes;
- Ramos: são rotulados com valores (atributos categóricos) ou com intervalos (atributos numéricos).

Para exemplificar a criação de uma árvore de decisão para a classificação, será utilizado o conjunto de dados sobre jogos apresentado na figura 11.10 para o treinamento do modelo.

Nr	Aspecto	Umidade	Vento	Jogo
1	Sol	Elevada	Fraco	Não
2	Sol	Elevada	Forte	Não
3	Nublado	Elevada	Fraco	Sim
4	Chuva	Elevada	Fraco	Sim
5	Chuva	Normal	Fraco	Sim
6	Chuva	Normal	Forte	Não
7	Nublado	Normal	Fraco	Sim
8	Sol	Elevada	Fraco	Não
9	Sol	Normal	Fraco	Sim
10	Chuva	Normal	Forte	Não
11	Sol	Normal	Forte	Sim
12	Nublado	Elevada	Forte	Sim
13	Nublado	Normal	Fraco	Sim
14	Chuva	Elevada	Forte	Não

Figura 11.10: Conjunto de dados sobre jogos.

O conjunto de dados apresentado na figura 11.10 é composto por quatorze linhas e cinco colunas. A coluna `Nr` representa o número de ordem do registro; a coluna `Aspecto` representa o aspecto climático que pode ser `Sol`, `Nublado` ou `Chuva`; a coluna `Umidade` representa a umidade relativa do ar que pode ser `Elevada` ou

`Normal` ; a coluna `Vento` representa a velocidade do vento que pode ser `Forte` ou `Fraco` ; e a coluna `Jogo` informa se houve ou não jogo.

Esse conjunto de dados foi coletado ao longo do tempo, porém cabe ressaltar que em projetos reais de classificação os conjuntos de dados utilizados possuem dezenas de colunas e milhares de linhas.

Assim, esse conjunto de dados rotulado registra se houve ou não jogo no passado de acordo com os valores dos atributos `Aspecto` , `Umidade` e `Vento` . Então, será criada uma árvore de decisão com esses atributos e respectivos valores que poderá ser utilizada para prever se haverá ou não jogo no presente ou no futuro a partir do conhecimento dos valores desses atributos.

A figura 11.11 apresenta o processo de geração de uma árvore de decisão a partir do conjunto de dados fornecido utilizando as bibliotecas do *scikit-learn*.

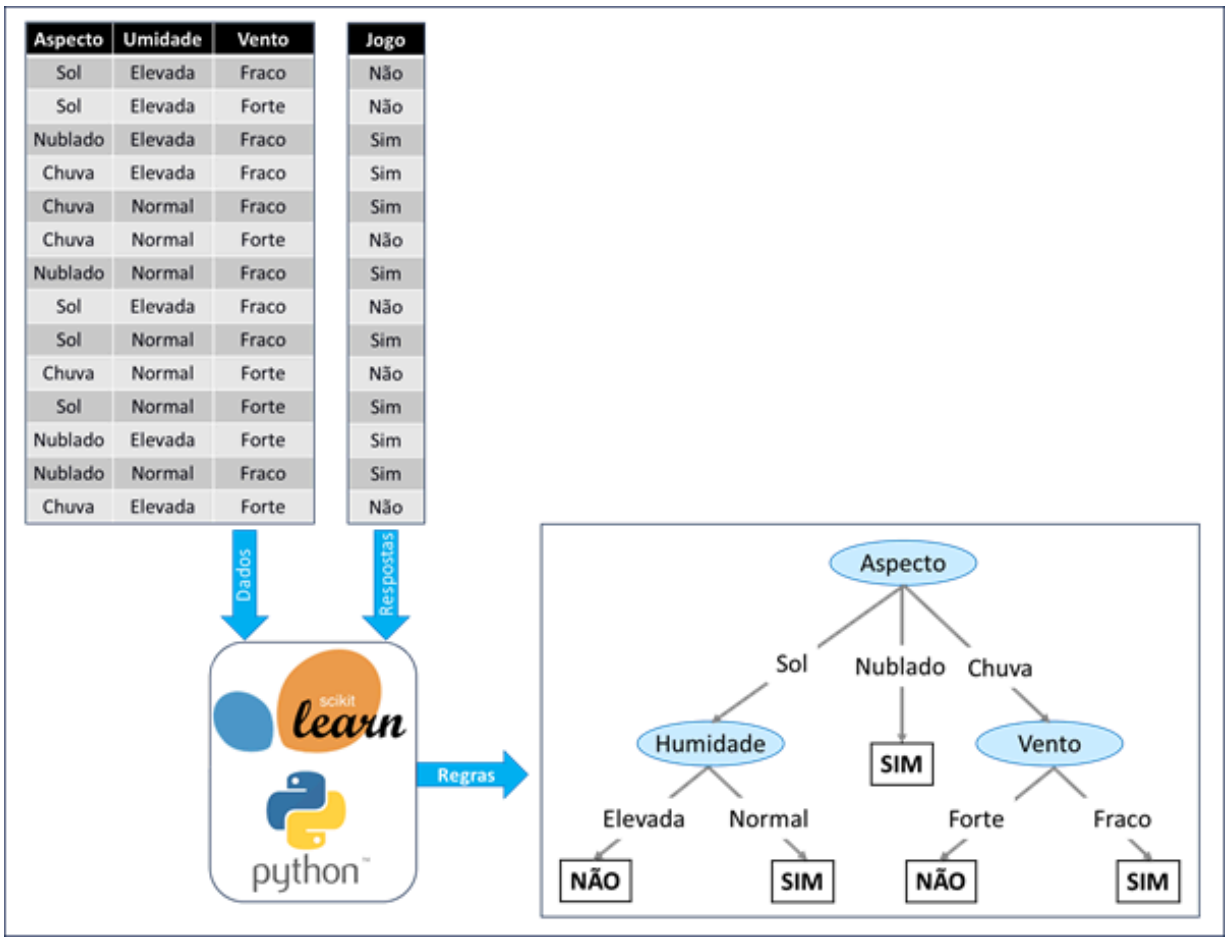


Figura 11.11: Geração de uma árvore de decisão utilizando o scikit-learn.

À esquerda e acima da figura 11.11, estão os atributos dos conjuntos de dados sobre jogos, ao lado está a indicação se houve jogo para cada um dos registros. Esses dados são processados pelo programa *scikit-learn* que gera a árvore de decisão localizada à direita da mesma figura.

Sendo assim, de acordo com a figura 11.11, o cientista de dados prepara um conjunto de dados composto por atributos climáticos relativos a dias passados (dados) e as respectivas ocorrências dos jogos (respostas). Após esse passo, ele configura os parâmetros do *scikit-learn* e submete esses dados ao processamento. Esse programa, por sua vez, gera uma árvore de decisão baseado nos dados climáticos e respostas fornecidos. A árvore de decisão gerada

permite a predição (estimativa) da ocorrência de novos jogos a partir dos atributos relacionados aos aspectos climáticos.

A disposição dos atributos nos nós da árvore é estabelecida pelo software utilizando a Entropia ou o Índice Gini. Ambos medem o nível de desordem dos dados, ou seja, a homogeneidade. A Entropia pode variar de 0 a 1, quanto mais próxima de 1 maior é a desordem dos dados, trazendo um menor ganho de informação para a predição de um dado futuro. Para o cálculo da Entropia usa-se a função logarítmica. O cálculo do Índice Gini é mais simples, ele também varia de 0 a 1, porém ambos os cálculos fogem do escopo desse livro, por isso não serão apresentados.

Por outro lado, os nós folhas representam as duas classes desse conjunto de dados, ou seja, "há jogo" ou "não há jogo". Essa árvore pode ser interpretada como um conjunto com as seguintes regras:

- SE (**Aspecto** = Sol) E (**Umidade** = Elevada) ENTÃO **Classe** = não há jogo.
- SE (**Aspecto** = Sol) E (**Umidade** = Normal) ENTÃO **Classe** = há jogo.
- SE (**Aspecto** = Nublado) ENTÃO **Classe** = há jogo.
- SE (**Aspecto** = Chuva) E (**Vento** = Forte) ENTÃO **Classe** = não há jogo.
- SE (**Aspecto** = Chuva) E (**Vento** = Fraco) ENTÃO **Classe** = há jogo.

Após a geração da árvore de decisão da figura 11.11, quando houver o interesse em saber se haverá jogo em um determinado dia, basta coletar as seguintes condições climáticas desse dia: `Aspecto`, `Umidade` e `Vento`. Após esse passo, será necessário percorrer a árvore de decisão comparando os valores dos atributos coletados com os atributos existentes nos nós da árvore e chegar ao nó folha mais apropriado. Esse nó folha representa a classe relativa aos atributos coletados, ou seja, essa classe indica se haverá ou não jogo. Cabe ressaltar que os modelos preditivos não garantem 100% de acerto, trata-se de estimativa.

Métodos baseados em otimização

A aprendizagem supervisionada baseada em otimização visa encontrar o melhor conjunto de parâmetros para um modelo preditivo. Nesse contexto, "otimização" refere-se ao processo de ajuste dos parâmetros do modelo através da minimização ou maximização de uma função objetivo. Esse ajuste é realizado por meio do treinamento do modelo utilizando conjuntos de dados rotulados, ou seja, com as classes predefinidas.

Os algoritmos de aprendizagem supervisionada baseados em otimização incluem classificação por funções matemáticas, Regressão Logística, Support Vector Machine (SVM), Deep Learning etc. Neste livro abordaremos apenas os algoritmos SVM e Deep Learning.

Support Vector Machine

Support Vector Machines (SVM) é um modelo de aprendizagem supervisionada que pode ser utilizado para classificação e análise de regressão. Nesta seção será abordado apenas o seu uso para a classificação.

Para explicar o algoritmo SVM, será utilizado o mesmo conjunto de dados apresentado na figura 11.2 que foi usado para a classificação utilizando o algoritmo K-NN, conforme apresentado na figura 11.3.

Diferentemente do algoritmo K-NN, que é um método baseado em distância, o algoritmo SVM é embasado na definição de um hiperplano que separa as classes do conjunto de dados usado para o treinamento. Hiperplanos são limites de decisão que auxiliam a separação dos dados em classes. A dimensão do hiperplano depende do número de atributos do conjunto de dados. Se o número de atributos for 2, o hiperplano será apenas uma linha. Se o número de atributos for 3, o hiperplano se tornará um plano bidimensional e

assim por diante. A figura 11.12 apresenta um exemplo de hiperplano representado por uma linha.

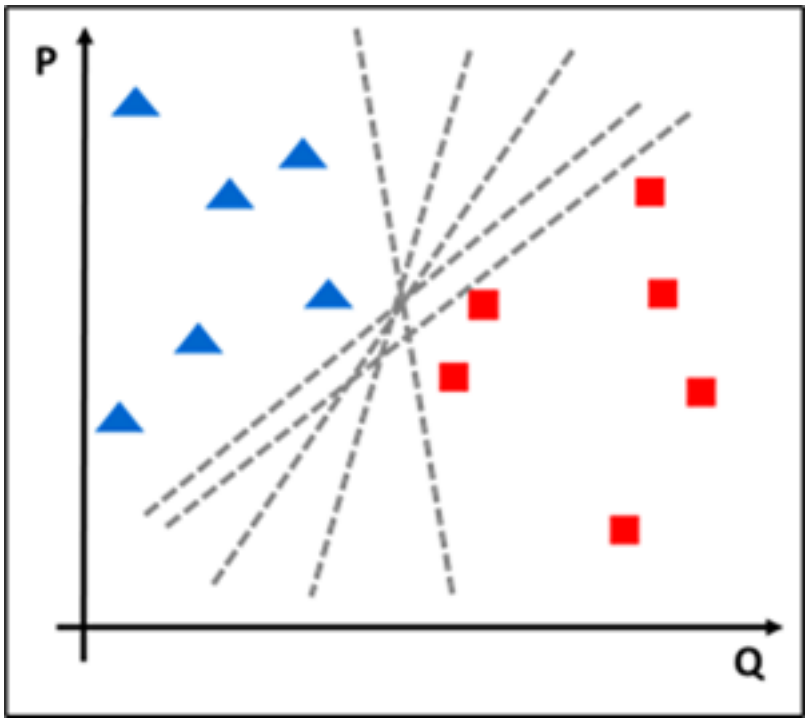


Figura 11.12: Divisão de dois conjunto de dados por meio de uma reta.

Como pode ser observado na figura 11.12, o conjunto de dados é composto por ocorrências formadas por duas dimensões, representadas pelos eixos `P` e `Q`, rotuladas em duas classes, representadas pelos triângulos e retângulos, linearmente separáveis. Então o hiperplano para a separação das duas classes será uma reta.

No entanto, existem inúmeras retas que podem separar as instâncias das duas classes existentes no conjunto de dados. Neste caso, a otimização consiste em encontrar a distância mínima entre dois itens de classes distintas e as margens que maximize a distância entre as classes, o hiperplano será a reta equidistante às duas margens encontradas. Os vetores de suporte estão relacionados às ocorrências do conjunto de dados que estão mais próximos dessa reta.

Como pode ser observado na figura 11.13, um conjunto de dados rotulados é apresentado ao algoritmo SVM. Esse conjunto de dados está organizado em duas classes: triângulos e retângulos. Após esse passo, o algoritmo fará várias operações por meio de vetores de suporte para encontrar os itens de classes distintas mais próximos e as margens do hiperplano (reta) que maximize a distância entre as classes.

O objetivo desse algoritmo é encontrar o hiperplano (reta) ideal de separação das duas classes que maximize a distância entre as margens, essas margens são equidistantes do hiperplano. Quanto maior for a distância entre as margens, mais clara será a separação entre as classes. A figura 11.13 apresenta o hiperplano e as margens encontradas para o conjunto de dados de treinamento.

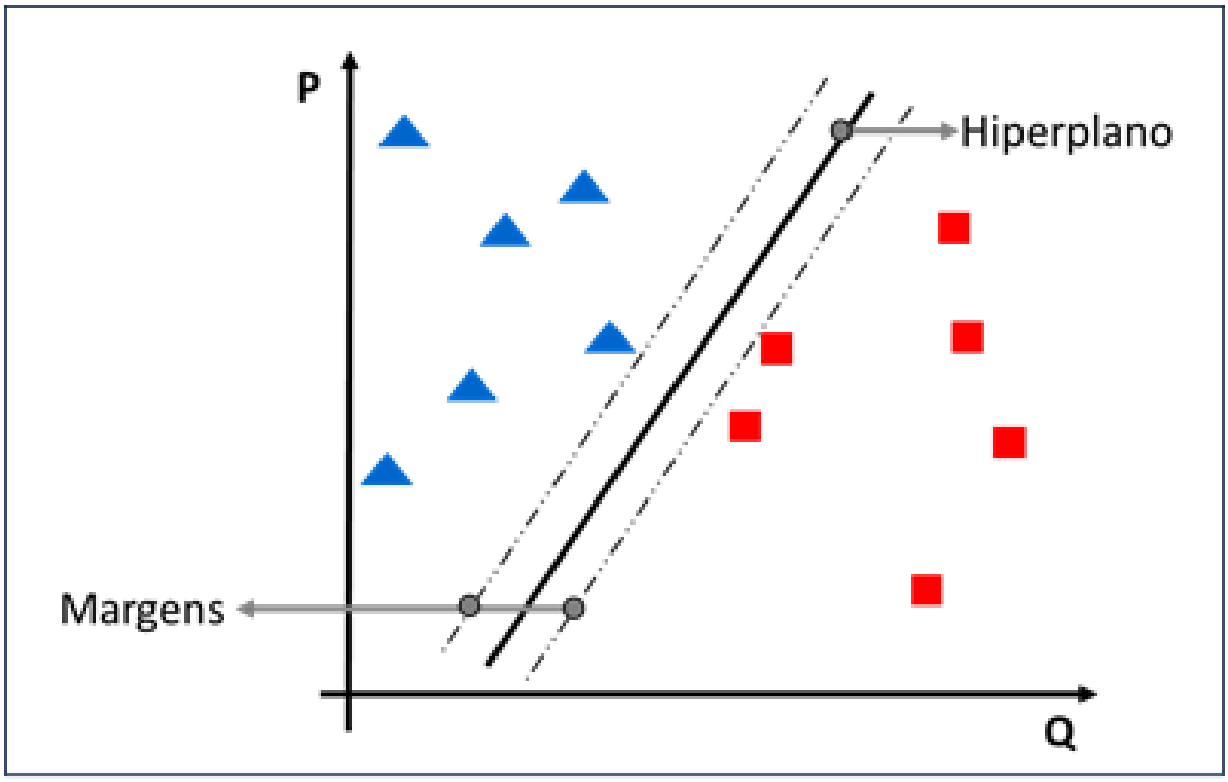


Figura 11.13: Classificação utilizando o algoritmo SVM.

Na figura 11.13, o hiperplano foi representado por uma reta, pois o conjunto de dados possui apenas duas dimensões e está organizado

em duas classes linearmente separáveis. Essa reta separa as duas classes do conjunto de dados com as margens mais distantes possível. Quando surgir um novo item de dado ainda não rotulado, basta verificar a sua posição no plano; se estiver acima da reta, esse elemento pertencerá à classe dos triângulos, caso contrário, pertencerá à classe dos retângulos.

Embora o algoritmo SVM tenha sido explicado nessa seção por meio de um conjunto de dados, cujos itens possuem apenas duas dimensões, organizados em duas classes linearmente separáveis, em projetos reais, esse algoritmo é muito utilizado para a classificação de conjunto de dados com várias dimensões, organizados em várias classes (multiclasses) e não linearmente separáveis, ou seja, uma reta não é suficiente para separar as classes.

Deep Learning

Deep Learning pode ser traduzido para "Aprendizado Profundo" que, na verdade, é proporcionado por Redes Neurais Profundas resultantes da evolução das Redes Neurais Artificiais (RNA). RNA é uma técnica computacional representada por um modelo matemático inspirado na estrutura neuronal do ser humano que assimila conhecimento através da experiência (treino). Consiste em uma rede de neurônios artificiais interconectados, que processam dados através de conexões ponderadas.

Cada neurônio artificial em uma RNA recebe entradas, que são ponderadas para gerarem saídas geralmente transformadas por meio das funções de ativação. A rede neural pode ter várias camadas de neurônios, incluindo uma camada de entrada, uma ou mais camadas ocultas, que são as camadas intermediárias, e uma camada de saída. O processo de treinamento de uma rede neural geralmente envolve ajustar os pesos das conexões entre os neurônios para que

a rede possa aprender a mapear corretamente as entradas para as saídas desejadas.

As RNAs têm sido muito utilizadas para atividades de classificação, processamento de linguagem natural, reconhecimento de imagem, diagnósticos e predição de doenças, carros autônomos, detecção de fraudes financeiras e várias outras. Elas são especialmente poderosas em lidar com problemas complexos e não lineares, onde métodos tradicionais tendem a ser menos eficazes.

Para exemplificar o funcionamento de uma rede neural, será apresentado o Modelo Perceptron, que foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt, inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts (DAS, 2022). O Perceptron é uma rede neural de camada única capaz de atuar como um classificador linear binário. Cabe ressaltar que, para ser considerada uma RNA, é necessário, no mínimo, um Perceptron de várias camadas. No entanto, para assimilar o funcionamento de uma rede neural, basta entender o processamento do Perceptron de camada única.

A figura 11.14 apresenta o Modelo Perceptron de uma camada.

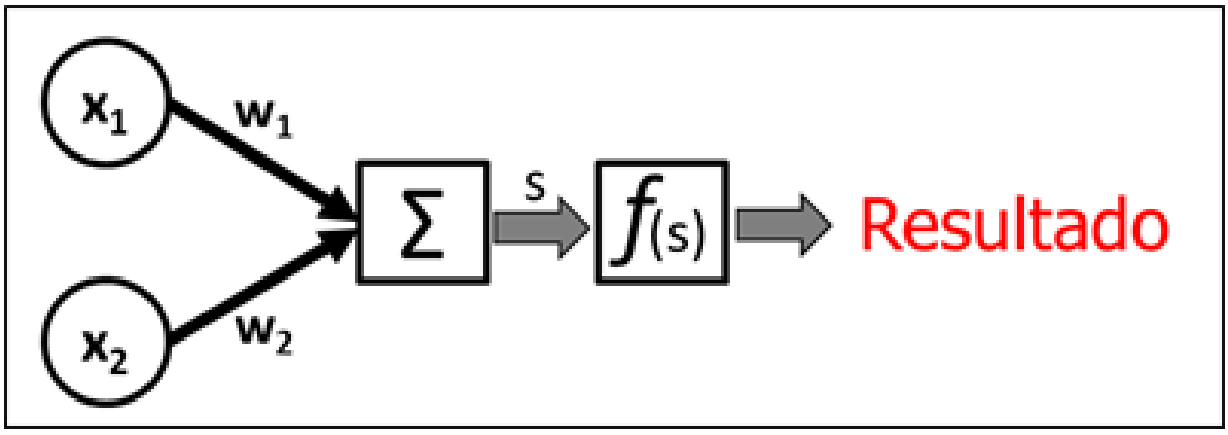


Figura 11.14: Modelo Perceptron de uma camada.

Conforme apresentado na figura 11.14, a operação de uma unidade de processamento Perceptron pode ser resumida da seguinte

maneira:

- Sinais são apresentados à entrada: `x1` e `x2`;
- Cada sinal é multiplicado por um número, ou peso, que indica a sua influência na saída da unidade: `w1` e `w2`;
- É feita a soma ponderada dos sinais que produz um nível de atividade: `s = (x1 * w1) + (x2 * w2)`;
- Este nível de atividade `s` passa pela função de ativação `f(s)` para verificar se o somatório excedeu um certo limite (*threshold*) ou não. O processamento dessa verificação determinará um resultado de saída.

O objetivo do treinamento de um modelo Perceptron é encontrar os valores dos pesos `w1` e `w2`.

A fim de exemplificar o funcionamento de uma rede neural, o Modelo Perceptron será utilizado para simular o operador booleano AND com duas entradas. A figura 11.15 apresenta a tabela *verdade* do exemplo a ser implementado. Essa tabela *verdade* já foi explicada na seção 8.2.

X1	X2	X1 AND X2
0	0	0
0	1	0
1	0	0
1	1	1

Figura 11.15: Tabela verdade do operador booleano AND.

A figura 11.16 apresenta o resultado da operação lógica `X1 AND X2` de cada uma das linhas da tabela verdade da figura 11.15 em um plano bidimensional, na qual o triângulo representa a saída `1` (verdadeiro) e o quadrado representa a saída `0` (falso), que podem

ser interpretadas como duas classes distintas. Como pode ser observado, essas duas classes são linearmente separáveis quando representadas em um plano bidimensional, ou seja, existe pelo menos uma reta capaz de separar as ocorrências das duas classes.

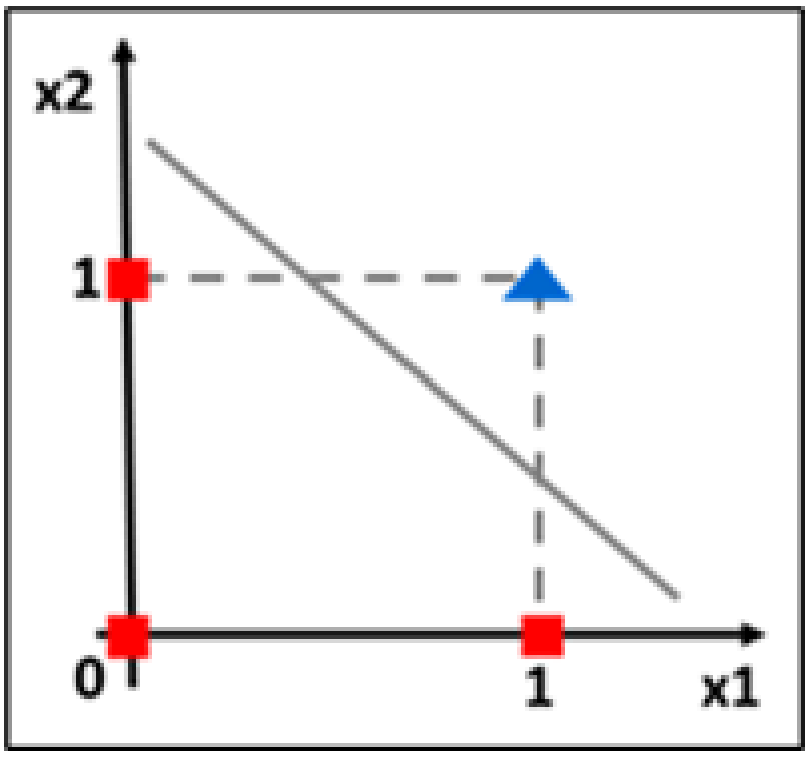


Figura 11.16: Os resultados do operador booleano AND representados em um plano bidimensional.

Conforme apresentado anteriormente, o conjunto de dados utilizado para o treinamento do Modelo Perceptron será composto por quatro ocorrências decorrentes da operação `AND` entre duas entradas. Essas ocorrências gerarão resultados que podem pertencer às classes `0` ou `1`. A classe `0` representa `falso` e a classe `1` representa `verdadeiro`. Essas classes podem ser separadas pela reta representada na figura 11.16, ou seja, os resultados acima da reta (apenas um neste caso) são da classe `1` e os que estão abaixo da reta são da classe `0`. As possíveis combinações com as respectivas classes são:

- (x1, x2) = (0,0) => 0 (quadrado)

- (x1, x2) = (0,1) => 0 (quadrado)
- (x1, x2) = (1,0) => 0 (quadrado)
- (x1, x2) = (1,1) => 1 (triângulo)

A tabela *verdade* da figura 11.15 será usada como conjunto de dados para treinar o Modelo Perceptron de camada única apresentado na figura 11.14 com a função de ativação `f(s)` apresentada na figura 11.17, o objetivo é encontrar os valores dos pesos `w1` e `w2`.

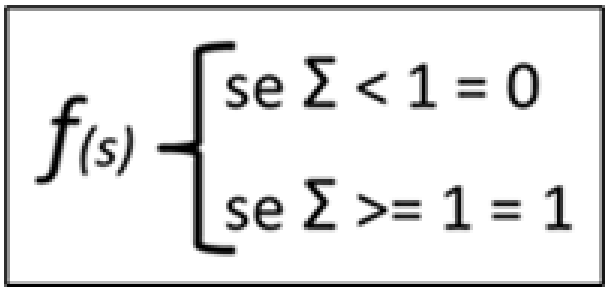


Figura 11.17: Função de ativação.

Como pode ser observado na figura 11.17, a função de ativação possui dois resultados possíveis:

- 0 => se o somatório das entradas multiplicadas pelos pesos for menor que 1.
- 1 => se o somatório das entradas multiplicadas pelos pesos for maior ou igual a 1.

Após a definição do conjunto de dados, das classes resultantes e da função de ativação, apresentada na figura 11.17, o próximo passo é iniciar o treinamento do modelo para encontrarmos os pesos `w1` e `w2` utilizando as quatro ocorrências da tabela verdade. A figura 11.18 apresenta a primeira rodada de treinamento atribuindo o valor 0 a esses pesos.

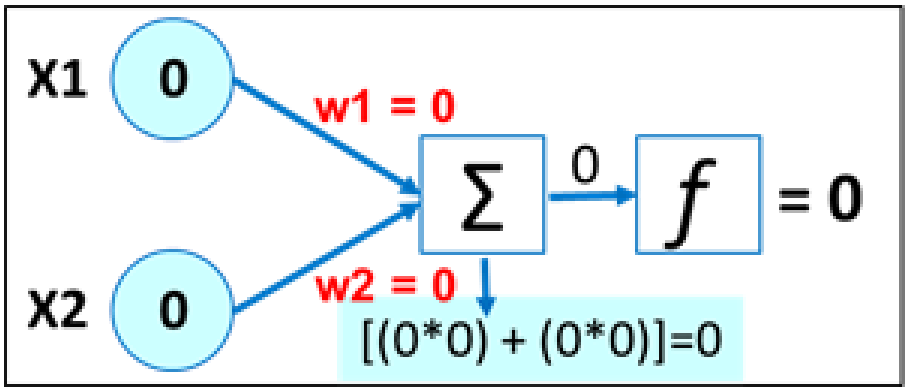


Figura 11.18: Treinamento do modelo para as entradas 0 e 0.

Como pode ser observado, a figura 11.18 aborda o treinamento do modelo para a primeira linha da tabela verdade do operador booleano `AND` apresentada na figura 11.15, no qual `x1 = 0` e `x2 = 0`.

Para essa etapa do treinamento, os pesos `w1` e `w2` receberam o valor 0. Assim, o somatório do valor de cada entrada multiplicado pelos pesos é igual à `(x1 * w1) + (x2 * w2) = (0 * 0) + (0 * 0) = 0`.

Testando o valor desse somatório na função de ativação da figura 11.17, o resultado será 0, pois o somatório é 0 e a função de ativação resulta em zero quando o valor da entrada é menor que 1. Nesse caso, o modelo Perceptron apresentou o valor correto na saída, pois sabe-se que `0 AND 0 = 0`.

A figura 11.19 aborda o treinamento do modelo para a segunda linha da tabela verdade do operador booleano `AND` apresentada na figura 11.15, no qual `x1 = 0` e `x2 = 1`.

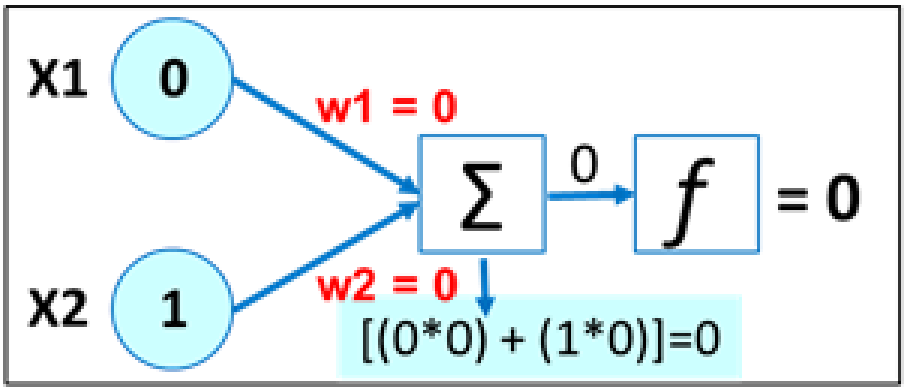


Figura 11.19: Treinamento do modelo para as entradas 0 e 1.

Para a etapa do treinamento apresentada na figura 11.19, os pesos `w1` e `w2` receberam o valor 0. Assim, o somatório do valor de cada entrada multiplicado pelos pesos é igual à `(x1 * w1) + (x2 * w2) = (0 * 0) + (1 * 0) = 0`.

Testando o valor desse somatório na função de ativação, o resultado será 0, pois o somatório é 0 e a função de ativação resulta em zero quando o valor da entrada é menor que 1. Nesse caso, o modelo Perceptron apresentou o valor correto na saída, pois sabe-se que `0 AND 1 = 0`.

A figura 11.20 aborda o treinamento do modelo para a terceira linha da tabela verdade do operador booleano `AND` apresentada na figura 11.15, no qual `x1 = 1` e `x2 = 0`.

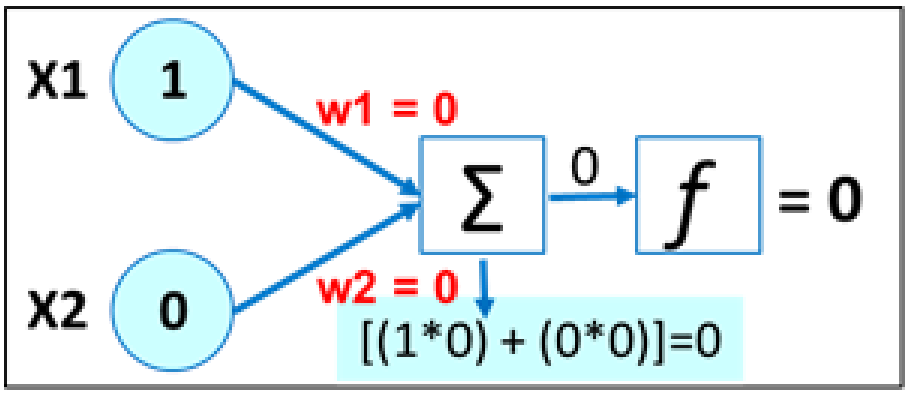


Figura 11.20: Treinamento do modelo para as entradas 1 e 0.

Para a etapa do treinamento apresentada na figura 11.20, os pesos `w1` e `w2` receberam o valor 0. Assim, o somatório do valor de cada entrada multiplicado pelos pesos é igual à `(x1 * w1) + (x2 * w2) = (1 * 0) + (0 * 0) = 0`.

Testando o valor desse somatório na função de ativação, o resultado será 0, pois o somatório é 0 e a função de ativação resulta em zero quando o valor da entrada é menor que 1. Nesse caso, o modelo Perceptron apresentou o valor correto na saída, pois sabe-se que `1 AND 0 = 0`.

A figura 11.21 aborda o treinamento do modelo para a quarta linha da tabela verdade do operador booleano `AND` apresentada na figura 11.15, no qual `x1 = 1` e `x2 = 1`.

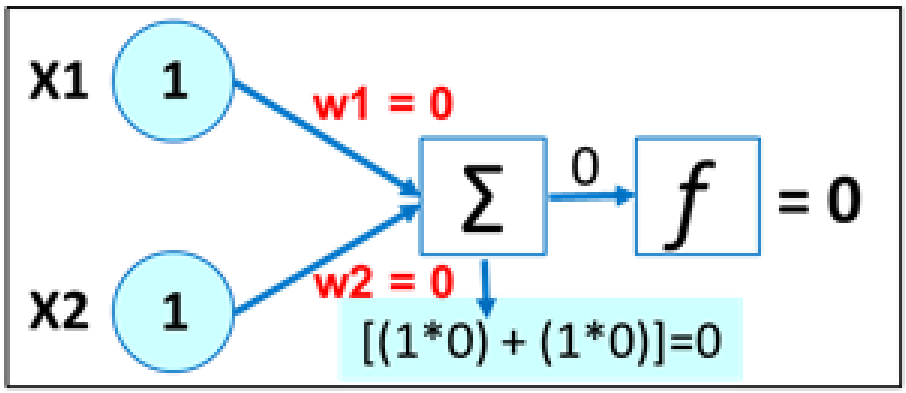


Figura 11.21: Treinamento do modelo para as entradas 1 e 1.

Para a etapa do treinamento apresentada na figura 11.21, os pesos `w1` e `w2` receberam o valor 0. Assim, o somatório do valor de cada entrada multiplicado pelos pesos é igual à `(x1 * w1) + (x2 * w2) = (1 * 0) + (1 * 0) = 0`.

Testando o valor desse somatório na função de ativação, o resultado será 0, pois o somatório é 0 e a função de ativação resulta em zero quando o valor da entrada é menor que 1. Nesse caso, o modelo Perceptron apresentou o valor incorreto na saída, pois sabe-se que `1 AND 1 = 1`.

Para dar continuidade ao processo de treinamento, é necessário calcular o erro total de cada passada, esse erro é calculado por meio de uma função de custo. A fim de simplificar o cálculo desse erro, pois em projetos reais esse cálculo é bem complexo, vamos considerar o erro total como o resultado da soma dos erros das predições de cada ocorrência, onde cada erro é calculado pela seguinte fórmula `erro = saída esperada – saída calculada`. Então, o erro total da primeira passada de treinamento será igual a:

- Para `0 AND 0` => erro = 0 – 0 = 0.
- Para `0 AND 1` => erro = 0 – 0 = 0.
- Para `1 AND 0` => erro = 0 – 0 = 0.
- Para `1 AND 1` => erro = 1 – 0 = 1.

Somando todos os erros apresentados anteriormente, teremos o `erro total = 1`. Considerando que o valor 1 seja um valor muito alto nesse contexto, então, será necessário atribuir novos valores aos pesos para esse erro ser minimizado.

Existem várias estratégias para a atribuição de novos valores aos pesos, elas podem ser sofisticadas ou não, mas sempre estão relacionadas a uma taxa de aprendizado. Aqui, omitiremos a fórmula para o cálculo dos novos pesos, para simplificarmos o aprendizado, simplesmente atribuiremos os novos valores aos pesos somando 0.1 ao peso utilizado na passada anterior do treinamento.

A figura 11.22 apresenta a segunda rodada de treinamento atribuindo o valor 0.1 a esses pesos.

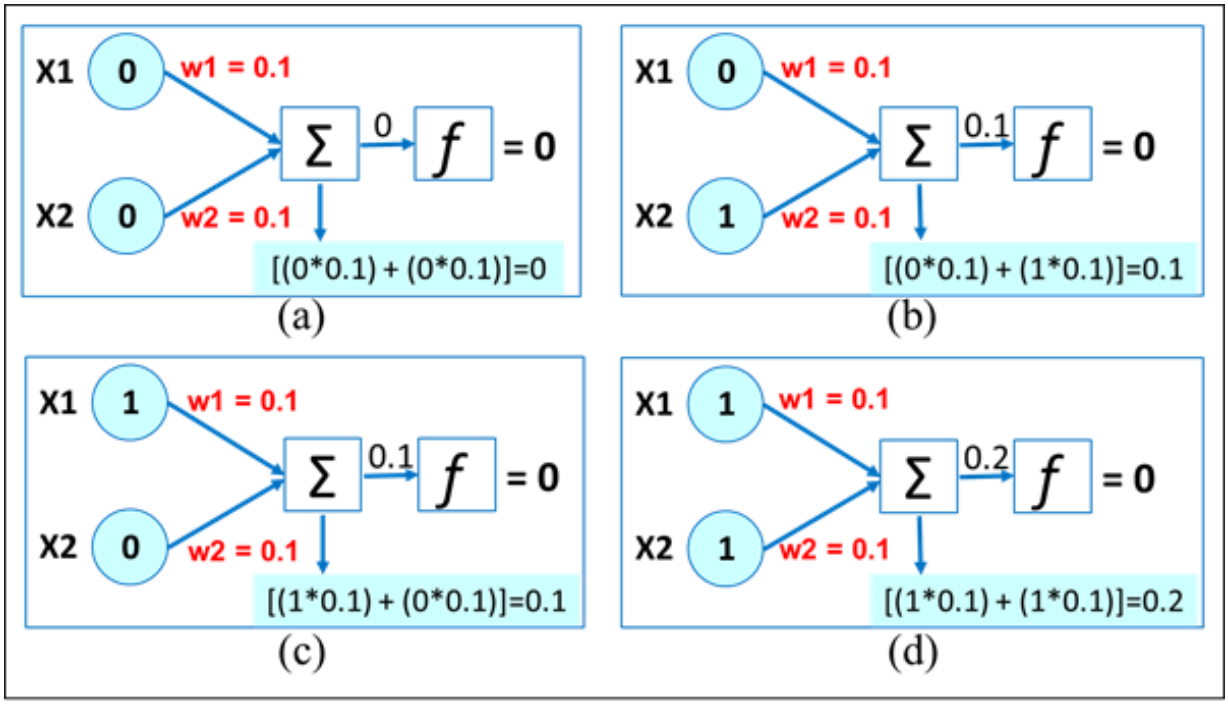


Figura 11.22: Treinamento do modelo para pesos com valor igual a 0.1.

Como pode ser observado na figura 11.22, o resultado da função de ativação para cada uma das ocorrências, cujos somatórios estão representados na parte inferior de cada modelo, será:

- Figura (a): Para `0 AND 0` => somatório = 0, saída da função de ativação = 0.
- Figura (b): Para `0 AND 1` => somatório = 0.1, saída da função de ativação = 0.
- Figura (c): Para `1 AND 0` => somatório = 0.1, saída da função de ativação = 0.
- Figura (d): Para `1 AND 1` => somatório = 0.2, saída da função de ativação = 0.

Na mesma figura, pode-se observar que o erro total da segunda rodada de treinamento será igual a:

- Para `0 AND 0` => erro = 0 – 0 = 0.
- Para `0 AND 1` => erro = 0 – 0 = 0.
- Para `1 AND 0` => erro = 0 – 0 = 0.

- Para `1 AND 1` => erro = 1 – 0 = 1.

Somando todos os erros apresentados anteriormente, teremos o erro total =1, ou seja, igual à rodada anterior de treinamento. Então, será necessário atribuir novos valores aos pesos para esse erro ser minimizado. Suponhamos que sejam feitas várias rodadas de treinamento até chegarmos ao valor 0.5 para os pesos.

A figura 11.23 apresenta a rodada de treinamento atribuindo o valor 0.5 a esses pesos.

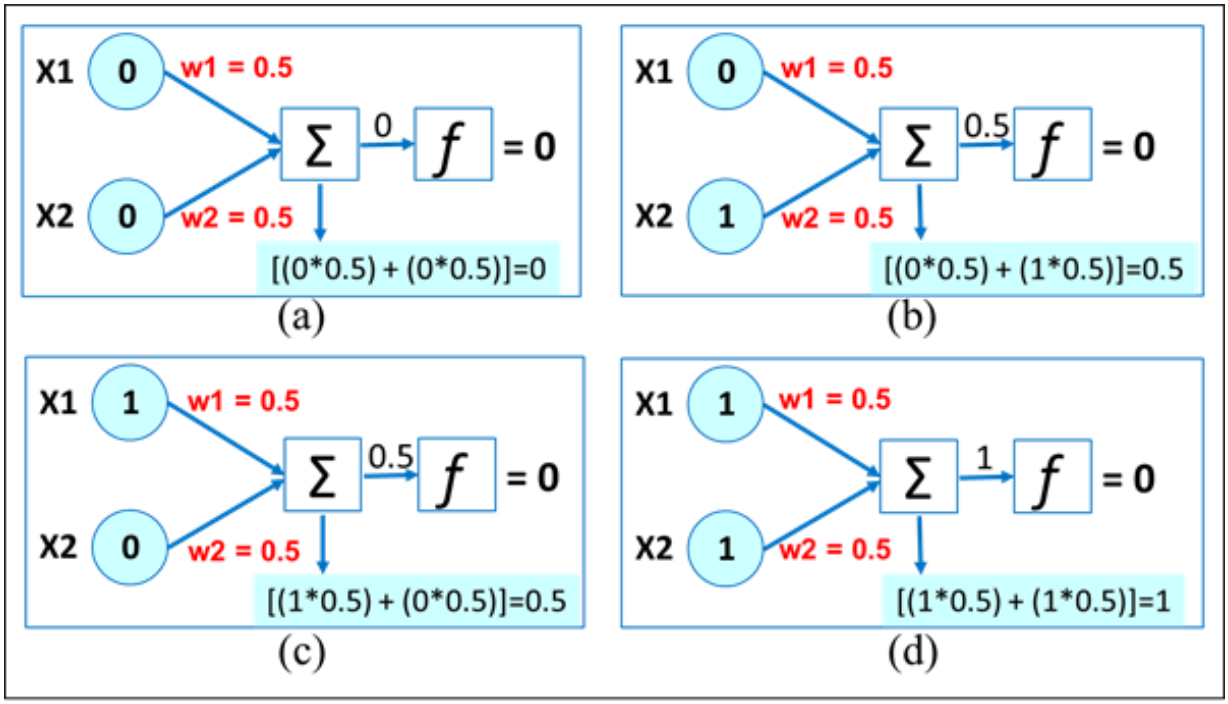


Figura 11.23: Treinamento do modelo para pesos com valor igual a 0.5.

Como pode ser observado na figura 11.23, o resultado da função de ativação para cada uma das ocorrências, cujos somatórios estão representados na parte inferior de cada modelo, será:

- Figura (a): Para `0 AND 0` => somatório = 0, saída da função de ativação = 0.
- Figura (b): Para `0 AND 1` => somatório = 0.5, saída da função de ativação = 0.

- Figura (c): Para `1 AND 0` => somatório = 0.5, saída da função de ativação = 0.
- Figura (d): Para `1 AND 1` => somatório = 1, saída da função de ativação = 1.

Na mesma figura, pode-se observar que o erro total da segunda rodada de treinamento será igual a:

- Para `0 AND 0` => erro = 0 – 0 = 0.
- Para `0 AND 1` => erro = 0 – 0 = 0.
- Para `1 AND 0` => erro = 0 – 0 = 0.
- Para `1 AND 1` => erro = 1 – 1 = 0.

Somando todos os erros apresentados anteriormente, teremos o erro total = 0, ou seja, chegou-se a um resultado excelente, muito difícil de ser atingido em projetos reais. Então, pode-se concluir que o modelo Perceptron pode representar a operação lógica `AND` entre duas sentenças quando atribuímos o valor 0.5 os pesos `w1` e `w2`, cabe ressaltar que existem milhares de combinações possíveis para os valores dos pesos `w1` e `w2` que chegariam ao mesmo resultado.

Para aprimorar o entendimento do modelo Perceptron, será apresentado de forma sucinta o seu uso para análise de crédito embasada no conjunto de dados sobre correntistas de uma instituição financeira apresentado na figura 11.24.

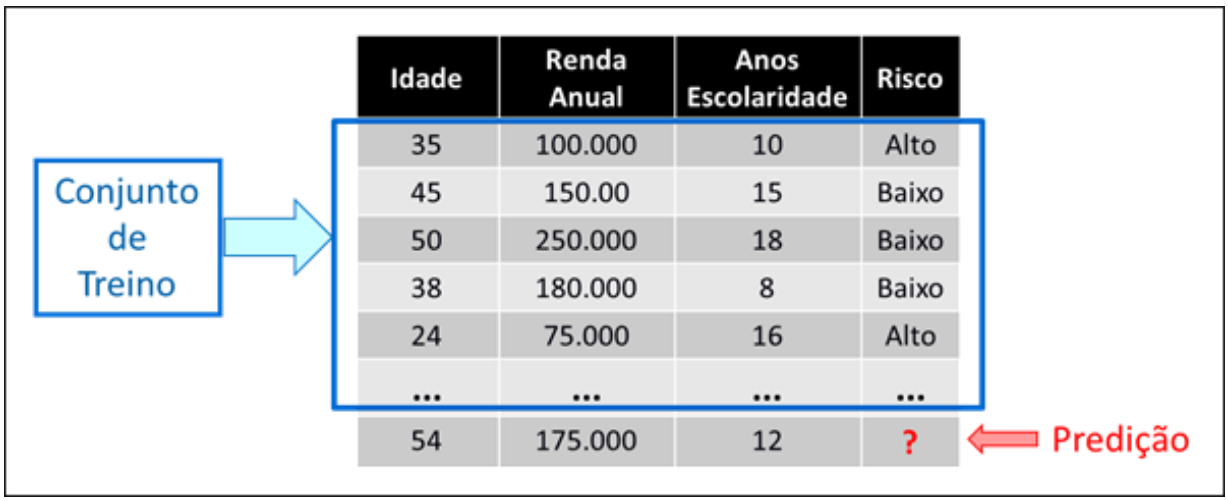


Figura 11.24: Conjunto de dados para análise de crédito.

A figura 11.24 apresenta um conjunto de dados com os atributos dos correntistas de uma instituição financeira, são milhares de linhas, embora só apareçam as 5 primeiras. Os atributos de cada linha são: `Idade`, `Renda Anual`, `Anos de Escolaridade` e `Risco`. Esse último atributo representa o risco de conceder crédito ao correntista, constituindo as duas possíveis classes de análise: o `Risco = Alto`, indica que o correntista é mau pagador, e o `Risco = Baixo`, indica o bom pagador. A última linha do conjunto de dados representa um potencial correntista que será avaliado pela instituição financeira para verificar a possibilidade de conceder ou não crédito. Essa análise será feita por meio do modelo da figura 11.25.

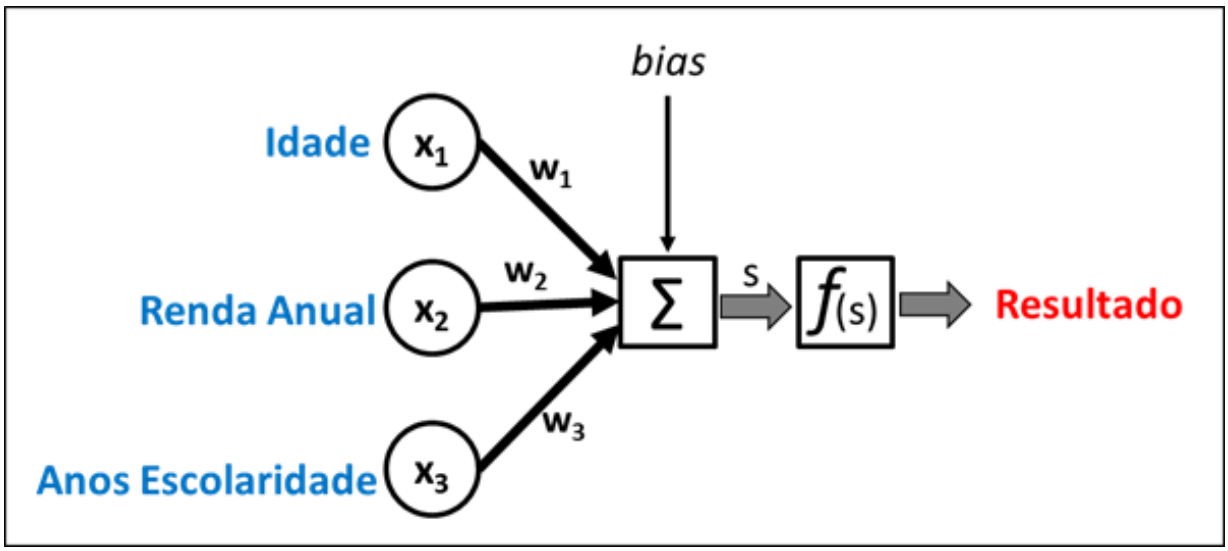


Figura 11.25: Modelo para análise de crédito.

O objetivo do modelo representado na figura 11.25 é proporcionar a predição indicando se um potencial correntista da instituição financeira analisada tende a ser um bom ou mau pagador, ou seja, se ele apresenta um risco alto ou baixo em pagar o crédito concedido. Esse potencial cliente é apresentado na última linha do conjunto de dados da figura 11.24.

O primeiro passo para fazer essa predição consiste em treinar o modelo da figura 11.25 usando o conjunto de dados da figura 11.24. Esse conjunto de dados possui três variáveis de entrada: `Idade`, `Renda Anual`, `Anos de Escolaridade`; e a variável `Risco`, que representa o resultado, ou seja, as duas possíveis classes, bom pagador e mau pagador.

O treinamento consiste em substituir `x1`, `x2` e `x3` pelos atributos de cada instância do conjunto de dados cujas classes já são conhecidas, ou seja, o resultado do modelo. O objetivo desse procedimento é encontrar os valores dos pesos `w1`, `w2`, `w3` e do *bias*. O *bias*, que pode ser traduzido para viés, é uma variável incluída ao somatório que será submetido à função de ativação, com o intuito de aumentar o grau de liberdade dessa função e, consequentemente, a capacidade de aproximação da rede.

Durante o treinamento, é possível atribuir valores aos pesos, ao *bias* e testar se as instâncias do conjunto de dados são classificadas nas classes corretas. Esse procedimento é repetido inúmeras vezes até se obter um resultado satisfatório, que depende do objetivo de cada projeto.

Após obter os valores dos pesos e do bias apropriados por meio do treinamento, é possível prever se potenciais clientes tendem a ser bons ou maus pagadores. Para essa predição, é necessário substituir as variáveis `x1`, `x2` e `x3` pelos atributos do cliente a ser classificado, neste caso `x1 = Idade = 54`, `x2 = Renda Anual = 175.000` e `x3 = Anos de Escolaridade = 12`. Em seguida, calcular o somatório `[(w1 * x1) + (w2 * x2) + (w3 * x3) + bias]`, utilizando os valores dos pesos e do *bias* encontrados, e testar o somatório encontrado na função de ativação para definir a qual classe pertence esse cliente. A partir do resultado, que pode ser bom pagador ou mau pagador, o responsável pela conceção de crédito poderá usá-lo para tomar a sua decisão de conceder ou não crédito ao novo cliente.

O modelo Perceptron é uma rede neural de camada única, para ser considerado uma Rede Neural Artificial é necessário um modelo Perceptron de várias camadas. Esse modelo é um classificador linear binário, ou seja, só pode ser utilizado para conjunto de dados onde haja apenas duas classes no qual essas duas classes sejam linearmente separáveis. Apesar de o modelo Perceptron ser bem simples, ele é muito útil para o entendimento de redes neurais.

Uma rede neural profunda é composta por uma camada de entrada, uma camada de saída e várias camadas intermediárias, também denominadas camadas ocultas. Essas redes neurais utilizam funções de ativação complexas e, teoricamente, podem trabalhar com qualquer tipo de entrada para alcançar qualquer tipo de resultado. Porém, elas precisam de muito mais treinamento do que outros métodos de Machine Learning, elas precisam de milhões de exemplos de dados de treinamento. Por isso, seu treinamento pode ser extremamente demorado e consumir demasiadamente recursos

computacionais. A figura 11.26 apresenta um exemplo de rede neural profunda.

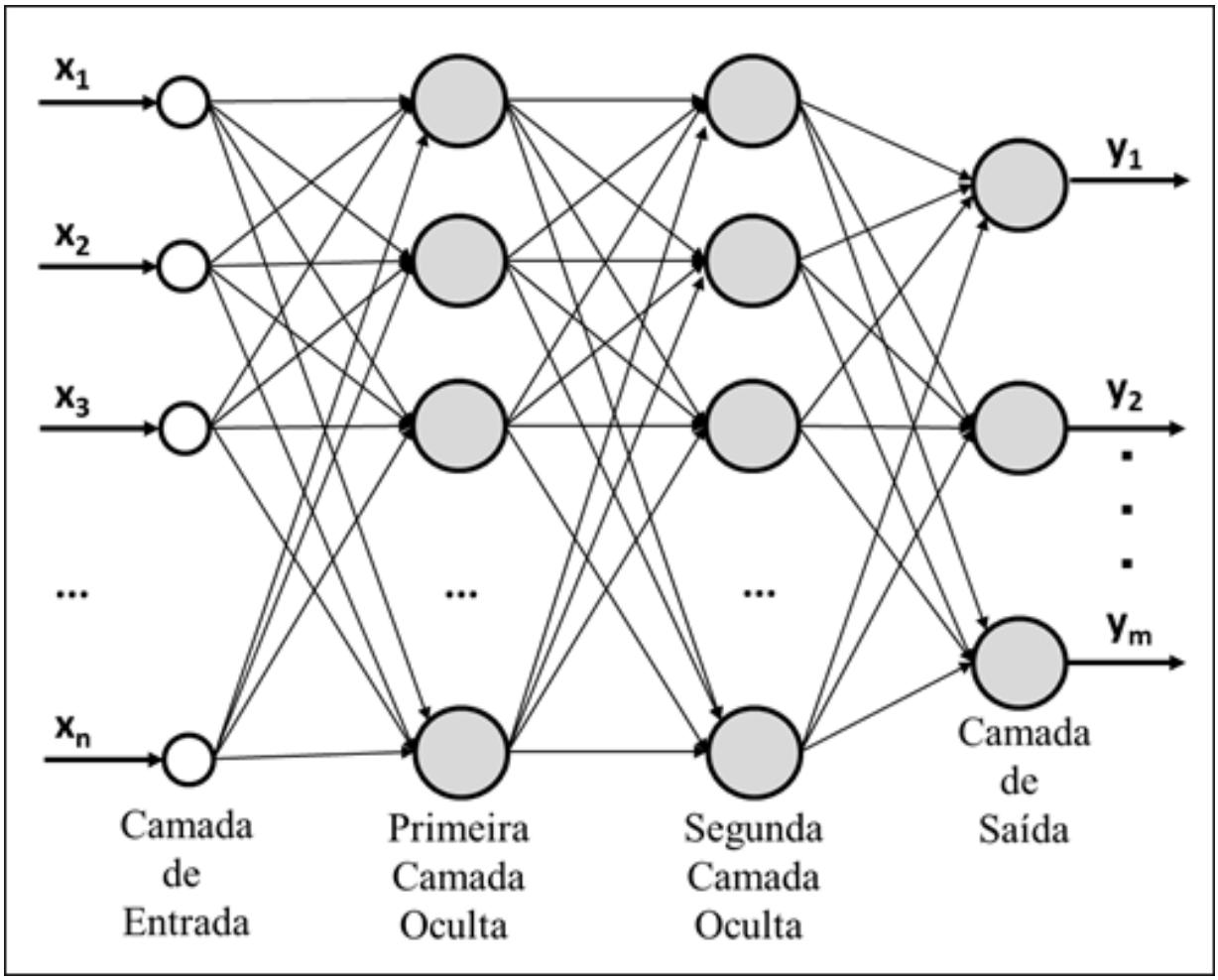


Figura 11.26: Rede neural profunda.

A figura 11.26 apresenta uma rede neural profunda com uma camada de entrada, uma camada de saída e duas camadas ocultas ou intermediárias. Cabe ressaltar que redes neurais profundas reais possuem várias camadas intermediárias, a figura apresenta apenas um exemplo didático.

Após a apresentação de vários métodos de classificação, na próxima seção será apresentado o método mais utilizado atualmente em projetos de Ciência de Dados para a predição, o método *ensemble* que, na verdade, consiste na junção de vários métodos.

Métodos *ensemble*

Os métodos *ensemble* são técnicas de aprendizado de máquina que combinam os resultados de um conjunto de modelos para obter um modelo preditivo mais eficiente. Esse conjunto pode ser composto por algoritmos distintos ou remodelações do mesmo algoritmo. O nome desses métodos tem origem na palavra francesa "*ensemble*" cujo significado pode ser "conjunto" ou "conjuntamente", entre outros. Assim, o resultado da predição é obtido por meio da combinação de vários métodos conjuntos que utilizam o aprendizado coletivo de vários modelos.

Um classificador *ensemble* consiste em um conjunto de classificadores treinados individualmente, denominados classificadores base, cujos resultados são combinados de alguma forma. A figura 11.27 apresenta a estrutura de um classificador *ensemble*.

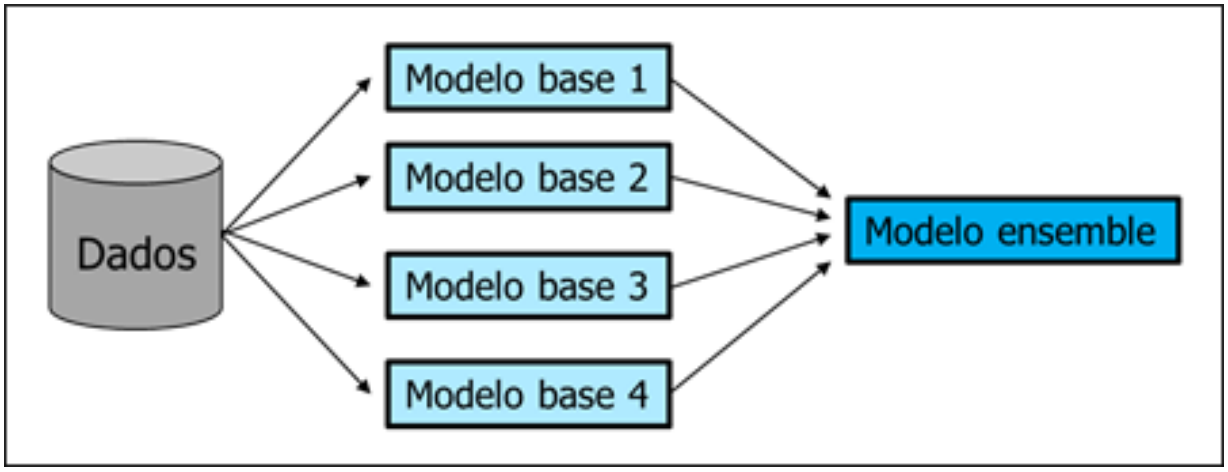


Figura 11.27: Estrutura de um classificador ensemble.

Como pode ser observado na figura 11.27, a predição realizada por um classificador *ensemble* consiste na combinação de várias predições de classificadores base.

Como exemplo de um classificador *ensemble*, pode-se destacar a *Random Forest*. *Random Forest* ou Floresta Aleatória é uma técnica

de aprendizado de máquina baseada em árvores de decisão. Consiste em construir diversas árvores de decisão durante o treinamento e combinar suas previsões por meio de votação para obter resultados mais precisos e robustos. Cada árvore é construída usando um subconjunto aleatório de recursos e um subconjunto aleatório de dados de treinamento. A figura 11.28 apresenta um exemplo da estrutura de uma *Random Forest*.

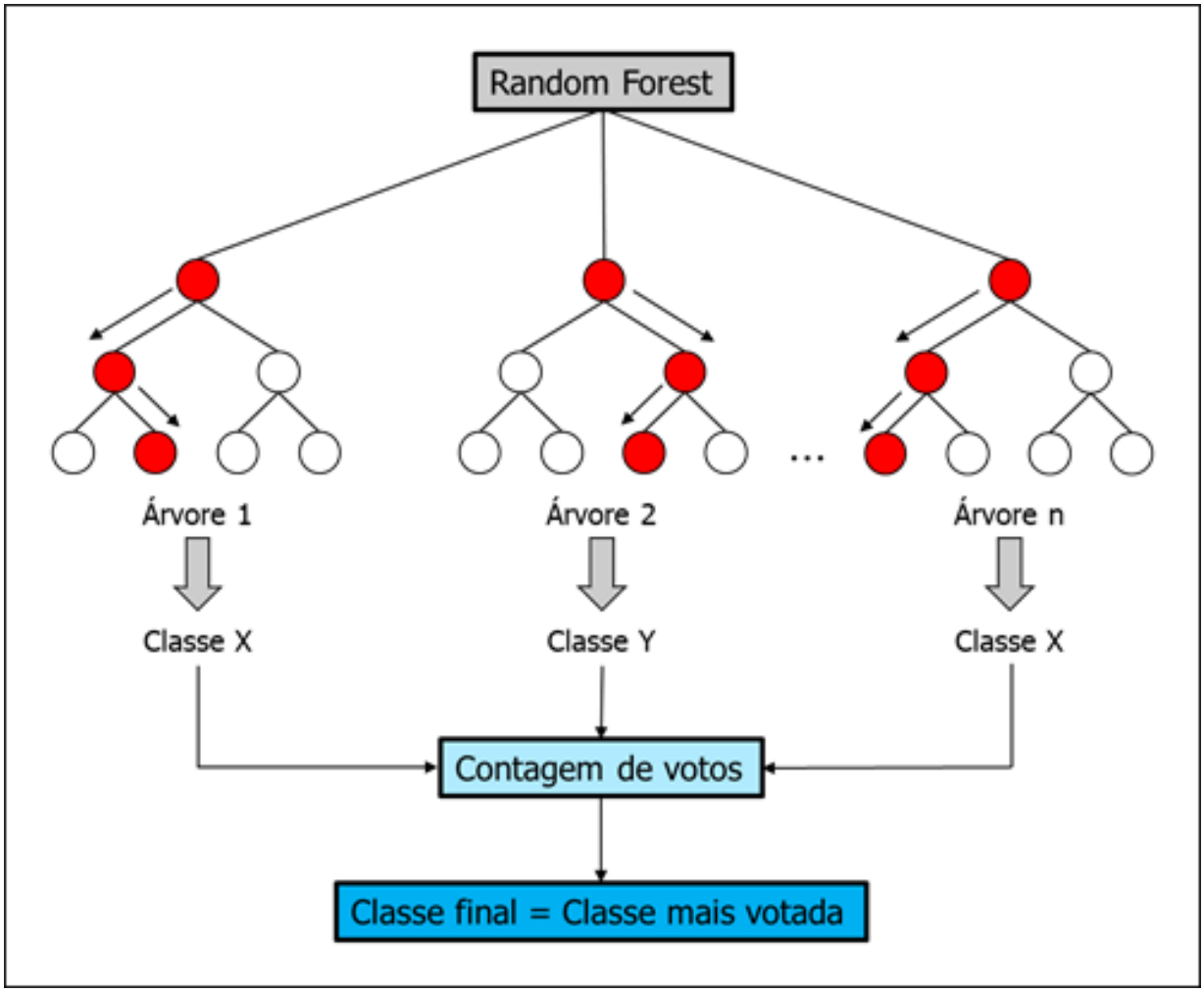


Figura 11.28: Estrutura de uma Random Forest.

Como pode ser observado na figura 11.28, a *Random Forest* foi obtida por meio do treinamento utilizando um conjunto de dados específico. Ela é composta por n árvores de decisão diferentes. Na figura estão representadas apenas a primeira, a segunda e a última

árvore. Todas elas têm como nós folhas apenas duas possíveis classes: a X e a Y.

Quando essa *Random Forest* é utilizada para prever se uma nova instância (ocorrência do conjunto de dados) pertence à classe X ou Y, é necessário percorrer cada uma das árvores que compõem a *Random Forest* comparando o valor de cada atributo dessa instância com os valores testados nos nós das árvores, como apresentado na seção 11.1.

O percurso de cada árvore está representado pelas setas e pelos círculos preenchidos da figura 11.28. Após a instância percorrer cada árvore, tem-se a respectiva classe resultante que pode ser interpretada como um voto. Então, após o percurso, faz-se a contagem dos resultados e a classe que obtiver o maior número de votos será a vencedora e considerada como o resultado da predição da instância utilizada.

No caso da figura 11.28, o resultado do percurso da árvore 1 foi a classe X, por outro lado, o resultado do percurso da árvore 2 foi a classe Y. Finalmente, o resultado do percurso da árvore n foi a classe X. Assim, se essa *Random Forest* fosse composta por apenas essas três árvores ($n = 3$), o resultado da votação seria 2 votos para a classe X e um voto para a classe Y. Então, o resultado global da predição seria a classe X.

Cabe ressaltar que *Random Forest* é apenas um dos exemplos de métodos *ensemble*, pois existem vários outros.

Os métodos *ensemble* constituem uma abordagem imprescindível para aperfeiçoar a performance dos modelos preditivos por combinarem vários modelos, por isso, são utilizados intensivamente em projetos de Ciência de Dados.

11.2 Regressão

A análise de regressão é um método de aprendizado supervisionado utilizado para calcular a relação entre duas ou mais variáveis, sendo uma delas dependente e as outras independentes. Com esse método, é possível criar um modelo que represente a relação entre as variáveis selecionadas e prever (estimar) valores com base no modelo. Embora haja vários tipos de regressão, será apresentada apenas a regressão linear simples para tornar a essência do conceito de análise de regressão mais claro.

A regressão linear simples é um caso especial da regressão linear em que há apenas uma variável independente que influencia a variável dependente. A equação da regressão linear simples consiste em uma equação de primeiro grau, que pode ser retratada por uma reta em um espaço bidimensional.

De forma simplificada, uma equação de primeiro grau pode ser caracterizada da seguinte forma:

$$y = ax + b$$

Onde:

- y = variável dependente;
- x = variável independente;
- a = o coeficiente de inclinação da reta;
- b = intercepto no eixo y, ou seja, o valor de y quando x é igual a zero.

O objetivo da regressão linear simples é estimar os coeficientes a e b da equação apresentada. Esse tipo de regressão é comumente utilizado em situações em que se deseja examinar a relação entre duas variáveis, por exemplo, a relação entre a distância percorrida por estudantes no caminho da escola e o tempo necessário para percorrê-la.

A figura 11.29 apresenta uma tabela com o nome dos estudantes, os valores das distâncias percorridas e o tempo necessário para percorrê-las.

Estudante	Distância (Km)	Tempo (min)
Ana	8	15
Beto	6	20
Caio	15	20
Eduardo	20	40
Lia	25	50
Júnior	11	25
Miguel	5	10
Nádia	32	55
Paula	28	35
Rita	20	30

Figura 11.29: Tabela com dados de estudantes.

Como pode ser observado, a tabela apresentada na figura 11.29 contém os dados de 10 estudantes. Segundo essa tabela, a estudante Ana percorreu 8 Km em 15 min, o estudante Beto percorreu 6 Km em 20 min e assim por diante. Um exemplo do uso da regressão linear simples seria tentar prever (estimar) quanto tempo um novo estudante, que ainda não tem seus dados registrados, demoraria para percorrer 22 Km, tendo como fonte de consulta apenas os dados contidos nessa tabela.

O primeiro passo para gerar um modelo capaz de fazer essa previsão (estimativa) é visualizar como esses dados estão dispostos

no espaço. A figura 11.30 apresenta o gráfico de dispersão dos dados apresentados na figura 11.29.

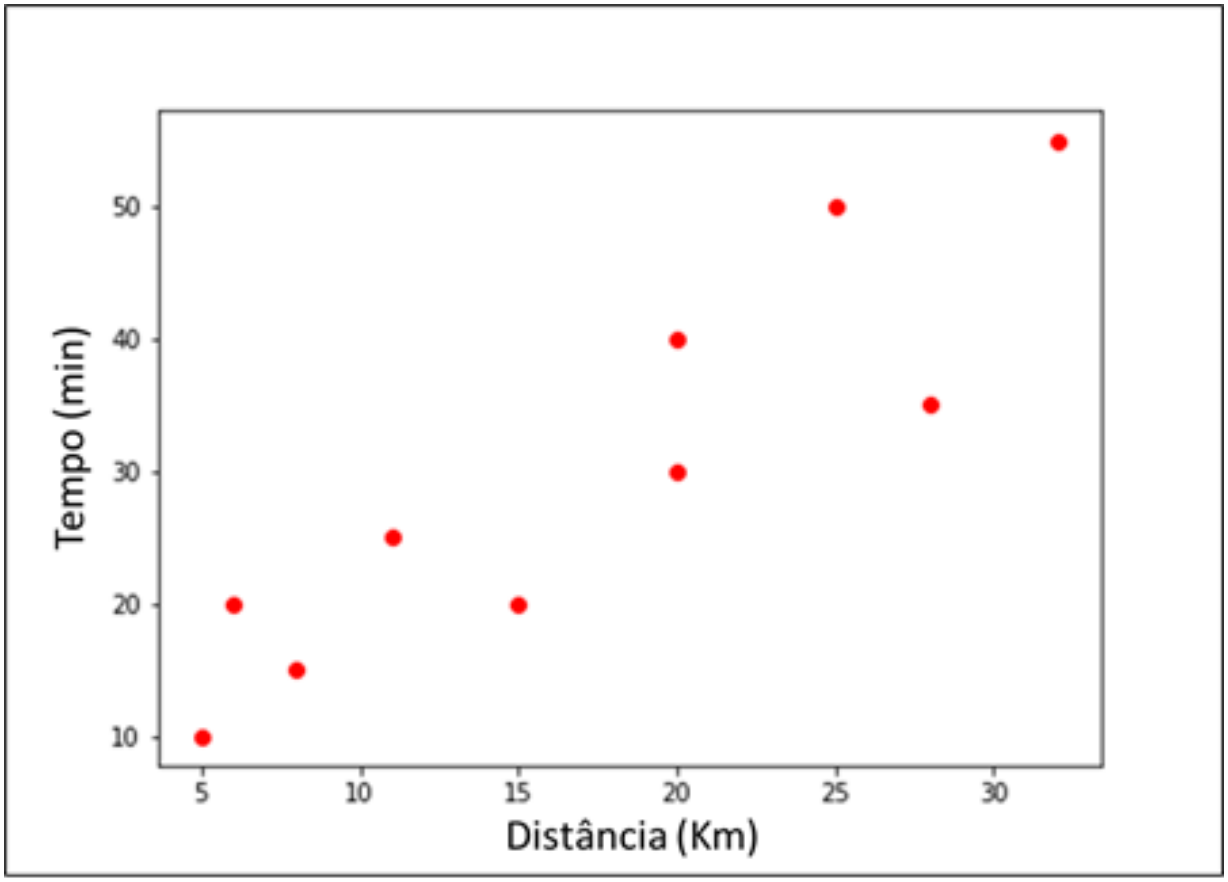


Figura 11.30: Gráfico de dispersão com dados de estudantes.

A figura 11.30 apresenta os dados da tabela da figura 11.29 no formato de pontos relativos às coordenadas `(x, y)` em um espaço bidimensional no qual o eixo `x` (horizontal) representa a distância percorrida, que é a variável independente, e o eixo `y` (vertical) representa o tempo necessário para percorrê-la, que é a variável dependente, ou seja, o valor a ser previsto (estimado).

Os pontos representados na figura possuem as seguintes coordenadas: `(8, 15)`, `(6, 20)`, `(15, 20)`, `(20, 40)`, `(25, 50)`, `(11, 25)`, `(5, 10)`, `(32, 55)`, `(28, 35)` e `(20, 30)`, que são as distâncias e os respectivos tempos para percorrê-las contidos na tabela da figura 11.29.

Como pode ser observado na figura 11.30, não existe uma reta que ligue exatamente todos os pontos, mas é possível encontrar retas que passem bem próximas a esses pontos. O objetivo é encontrar a melhor reta que represente uma generalização da relação entre as variáveis distância e tempo. A figura 11.31 apresenta algumas dessas possíveis retas.

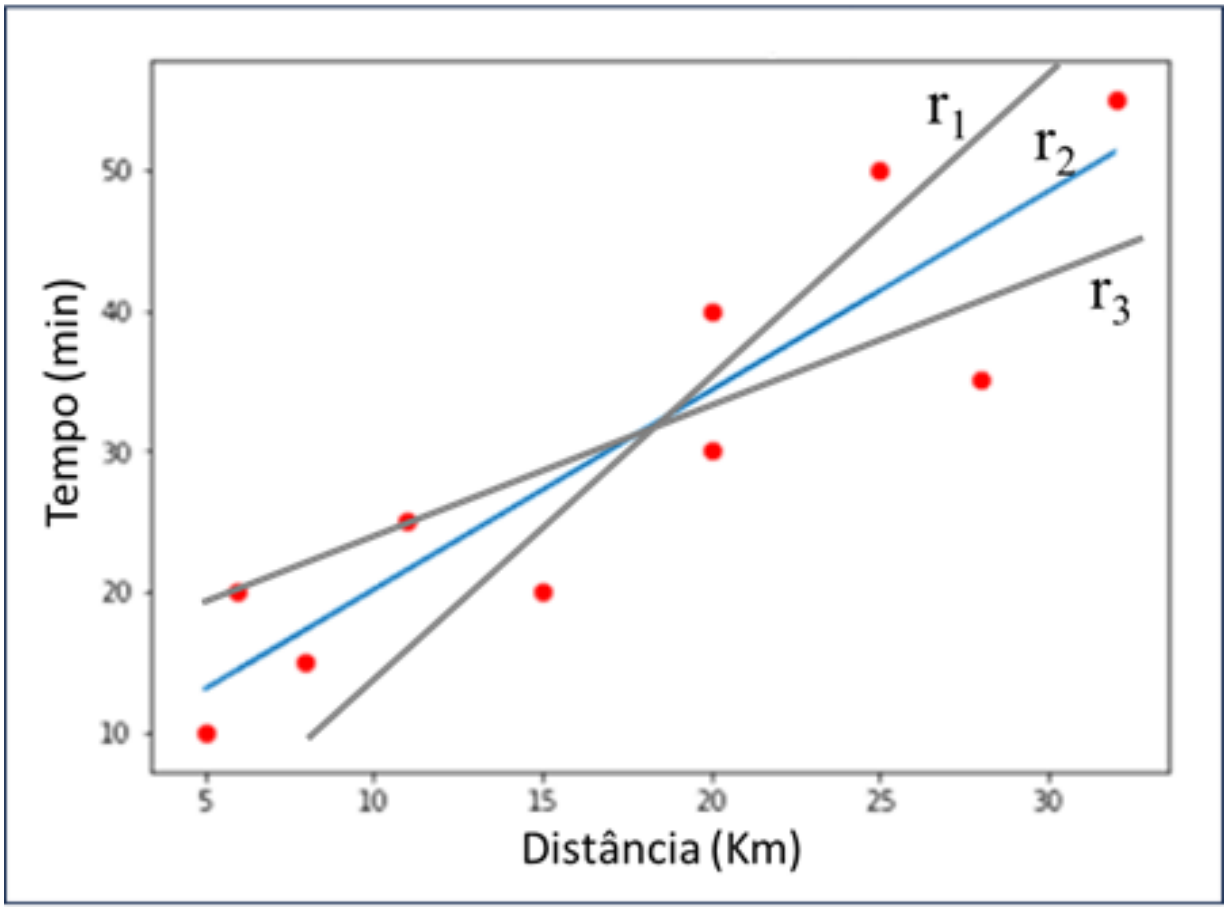


Figura 11.31: Retas próximas às coordenadas distância x tempo.

A figura 11.31 apresenta algumas das possíveis retas que passam próximas às coordenadas relativas à distância e ao tempo representadas nos eixos horizontal e vertical, respectivamente. Mas como encontrar a melhor reta que represente esses pontos?

O problema a ser resolvido com a regressão linear simples é encontrar a equação de uma reta que minimize o erro durante a previsão, ou seja, que possua a menor diferença entre os valores

reais e estimados. Os valores reais estão na tabela da figura 11.31 e os valores previstos são os pontos pertencentes a essa reta que pode ser representada pela equação $y' = ax + b$, onde:

- y' (y linha) = valor estimado para variável dependente;
- x = variável independente;
- a = o coeficiente de inclinação da reta;
- b = intercepto no eixo y' (y linha), ou seja, o valor de y' (y linha) quando x é igual a zero.

Então, a estratégia para definir a melhor reta é encontrar os valores de a e b para a equação apresentada, com o objetivo de proporcionar a menor soma das diferenças entre os valores reais e estimados para a variável dependente, que é o tempo, a partir do momento que se tenha o valor da variável independente, que é a distância. A figura 11.32 apresenta a diferença entre o valor estimado y' (um ponto na reta) e valor real y (20 min) dos tempos para a distância de 15 Km.

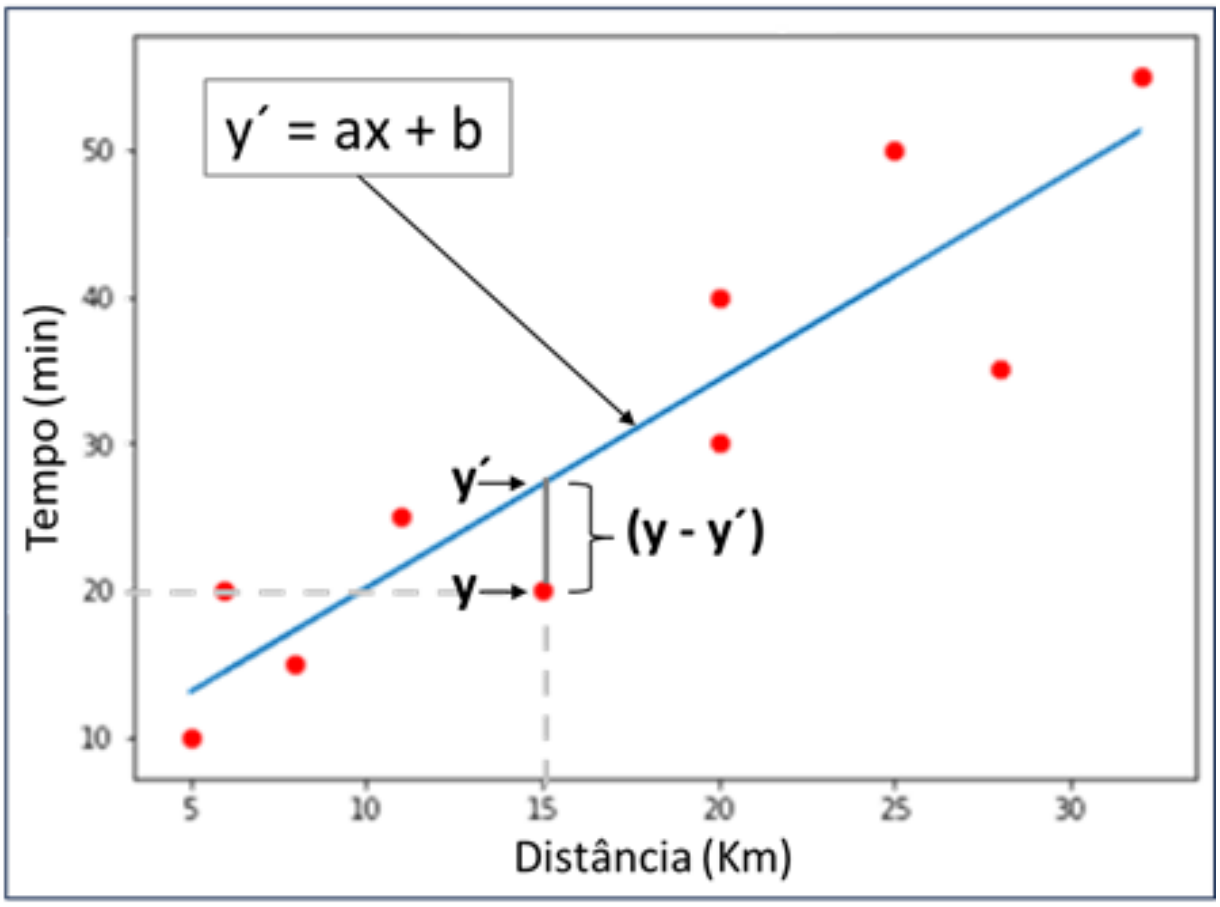


Figura 11.32: Diferença entre o valor estimado y´ e valor real y dos tempos para a distância de 15 Km.

Como pode ser observado na figura 11.32, quando a distância a ser percorrida é 15 Km (x), são necessários 20 min (y) para percorrê-la. No entanto, gostaríamos de prever qual seria o tempo necessário para um aluno percorrer 22 Km. Como seria possível estimar esse tempo?

O objetivo da regressão linear é gerar um modelo generalizado para estimar o valor do tempo, dada uma distância x, por meio da reta $y´ = ax + b$, onde $y´$ (y linha) é o valor estimado do tempo para percorrer a distância x. Para chegarmos à equação da reta, é preciso calcular os valores de a e b. O ponto de partida para encontrar os valores de a e b consiste em calcular o erro da previsão que é igual a diferença entre o valor de y (tempo real) e y

´ (tempo estimado, obtido por meio da reta) para as distâncias com os respectivos tempos conhecidos (figura 11.31), assim:

- erro em relação ao tempo para cada valor de distância = `(y – y´)`;
- como `y´= ax + b`;
- então, substituindo `y´` por `ax + b`;
- temos que `(y – y´)` = `(y – (ax + b))`;
- como `(y – (ax + b))` = `(y – ax - b)`;
- então, diferença = `(y – ax - b)`.

Assim, o erro total dos valores previstos em relação aos valores reais é calculado pela soma do erro de cada um dos dez pontos já conhecidos. Porém esse erro poderá ser positivo ou negativo, por isso eleva-se esse erro ao quadrado para transformar todos os números negativos em positivos. Então, o somatório de todos os erros ao quadrado é calculado pela fórmula da figura 11.33.

$$\sum_{i=1}^{n} (y_i - y_i')^2$$

Figura 11.33: Soma de todos os erros ao quadrado.

Na figura 11.33, o `i` variará de 1 a n, onde o valor máximo de n será 10, tendo em vista que o conjunto de dados utilizado possui 10 registros. Assim, a expressão representa o somatório das diferenças do tempo real e o tempo estimado dos 10 pontos fornecidos na tabela da figura 11.29.

Para calcularmos os valores de `a` e `b` da reta de regressão, podemos utilizar o **Método dos Mínimos Quadrados (MMQ)**, esse método procura minimizar o erro total ao quadrado

apresentado na figura 11.33. Cabe ressaltar que é possível substituir y' pela equação $ax + b$. A figura 11.34 apresenta essa substituição na expressão da figura 11.33.

$$\sum_{i=1}^{n} (y_i - ax_i - b)^2$$

Figura 11.34: Substituição de y´ pela equação "ax + b".

Ao encontrarmos o valor mínimo do somatório apresentado na figura 11.34, utilizando recursos matemáticos, que não serão apresentados, pois fogem do objetivo deste livro, encontraremos as fórmulas para a e b apresentadas na figura 11.35.

$$a = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$b = \bar{y} - a\bar{x}$$

$\bar{y}$ = média dos valores de y.
$\bar{x}$ = média dos valores de x.

Figura 11.35: Fórmula para o cálculo dos coeficientes "a" e "b" da equação de regressão linear.

Como pode ser observado na figura 11.35, a fórmula para o cálculo de `a` contém apenas valores relativos aos dez pontos conhecidos, apresentados na figura 11.29, são eles: as distâncias (`x`), os tempos (`y`), as médias das distâncias (`x barra`) e a média dos tempos (`y barra`). A tabela da figura 11.36 foi criada para facilitar o cálculo de `a` .

1	2	3	4	5	6	7
Estudante	Distância (x)	Tempo (y)	(x - media_x)	(x - x_media)^2	(y - media_y)	(x - media_x) * (y - media_y)
Ana	8	15	-9	81	-15	135
Beto	6	20	-11	121	-10	110
Caio	15	20	-2	4	-10	20
Eduardo	20	40	3	9	10	30
Lia	25	50	8	64	20	160
Júnior	11	25	-6	36	-5	30
Miguel	5	10	-12	144	-20	240
Nádia	32	55	15	225	25	375
Paula	28	35	11	121	5	55
Rita	20	30	3	9	0	0
Média →	**17**	**30**	**Somatório →**	**814**	**Somatório →**	**1.155**

Figura 11.36: Tabela para o cálculo do coeficiente "a" da equação de regressão linear.

As colunas da tabela da figura 11.36 apresentam os seguintes dados:

- Coluna 1: nome dos estudantes;
- Coluna 2: o valor de cada distância percorrida pelos estudantes e a média desses valores (última linha);
- Coluna 3: o valor de cada tempo necessário para os estudantes percorrerem cada distância e a média desses valores (última linha);
- Coluna 4: o valor de cada distância menos a média dessas distâncias;
- Coluna 5: o valor de cada distância menos a média dessas distâncias elevado ao quadrado e o somatório desses valores (última linha);
- Coluna 6: o valor de cada tempo menos a média desses tempos;

- Coluna 7: o produto do valor de cada linha da coluna 4 vezes o valor de cada linha da coluna 6 e o somatório desses valores (última linha).

Assim podemos concluir que para calcularmos o coeficiente `a` da reta da regressão linear utilizando a fórmula apresentada na figura 11.35, basta dividirmos a última linha da coluna 7 da tabela da figura 11.36 pela última linha da coluna 5 da mesma tabela, ou seja, `1,155 / 814 = 1,4189`.

Para obtermos o valor do coeficiente `b`, segundo a fórmula apresentada na figura 11.35, basta calcularmos a média dos tempos (última linha da coluna 3) e subtrairmos o valor de `a` multiplicado pelo valor da média das distâncias (última linha da coluna 2), ou seja, `b = 30 - (1,4189 x 17) = 5,8787`. Assim, a equação da reta de regressão será `y' = 1,4189x + 5,8787`, onde:

- `y' (y linha)` = tempo estimado;
- `x` = distância.

A figura 11.37 apresenta um resumo dos passos seguidos para gerar a reta de regressão linear para o conjunto de dados fornecido utilizando o Método dos Mínimos Quadrados.

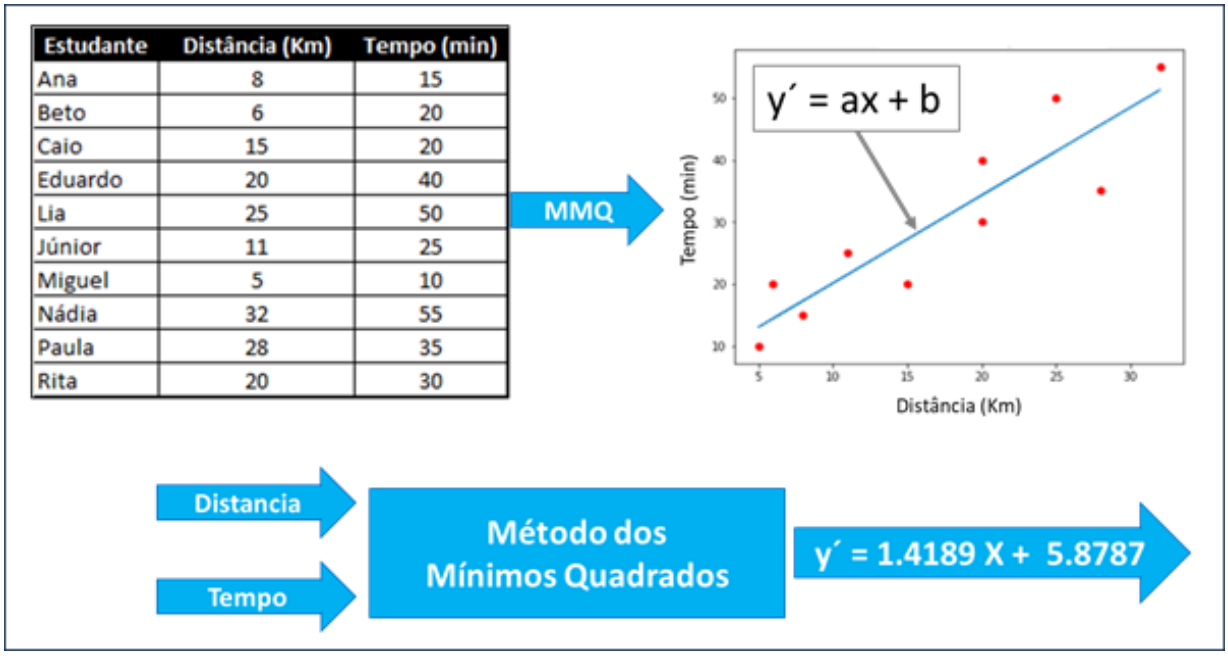


Figura 11.37: Resumo da aplicação do Método dos Mínimos Quadrados.

Como pode ser observado na figura 11.37, inicialmente foi fornecido um conjunto de dados com dez registros com o nome de estudantes, uma variável independente (x), relativa à distância que eles percorrem, e uma variável dependente (y), relativa ao tempo necessário para percorrer essa distância.

Foi aplicado o MMQ a esses dados e chegou-se à equação da reta de regressão linear $y' = 1{,}4189x + 5{,}8787$. Como pode ser visto na figura 11.37, essa reta não passa exatamente pelos pontos do conjunto de dados, mas ela proporciona o erro mínimo, possibilitando uma estimativa aproximada dos tempos para percorrer novas distâncias, por exemplo, o tempo necessário para um estudante percorrer 22 Km.

Para calcularmos esse tempo, basta substituirmos o x da equação pelo valor 22. Assim teremos $y' = 1{,}4189\ (22) + 5{,}8787 = 37{,}1$. A figura 11.38 apresenta essa predição no gráfico da reta de regressão.

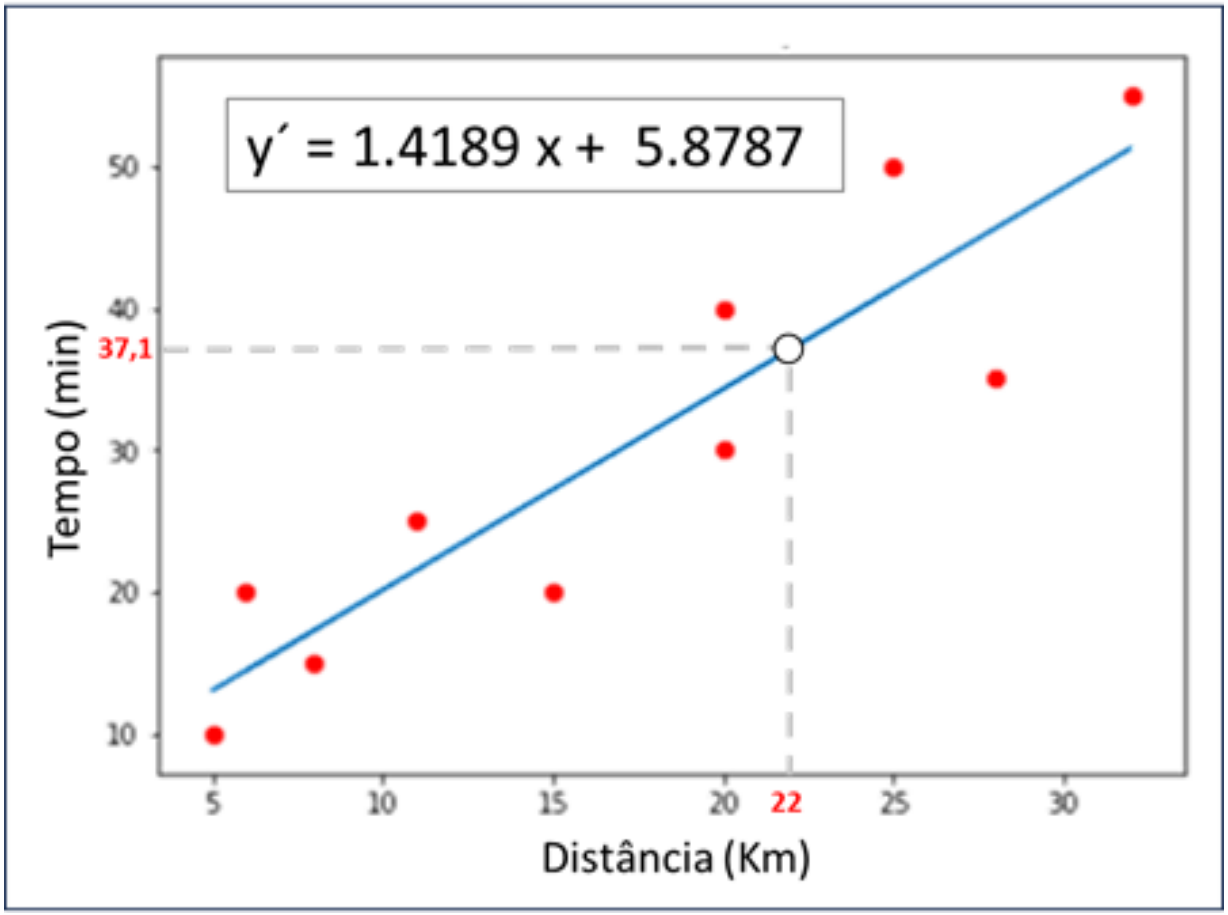


Figura 11.38: Estimativa do tempo necessário para um estudante percorrer 22 Km.

Cabe ressaltar, mais uma vez, que a predição, na verdade, é uma estimativa proporcionada por um modelo generalizado, que neste caso consiste na equação de uma reta. Outro aspecto importante a ser lembrado é que o MMQ é apenas um dos métodos para se obter a reta de regressão linear, pois existem outros.

Apesar da análise de regressão ser um recurso muito útil para estimar valores, nem sempre é uma estratégia viável para problemas de predição. Para ser utilizada, é necessário que haja correlação positiva ou negativa entre as variáveis estudadas. A figura 11.39 apresenta os diagramas de dispersão para os três tipos de correlação.

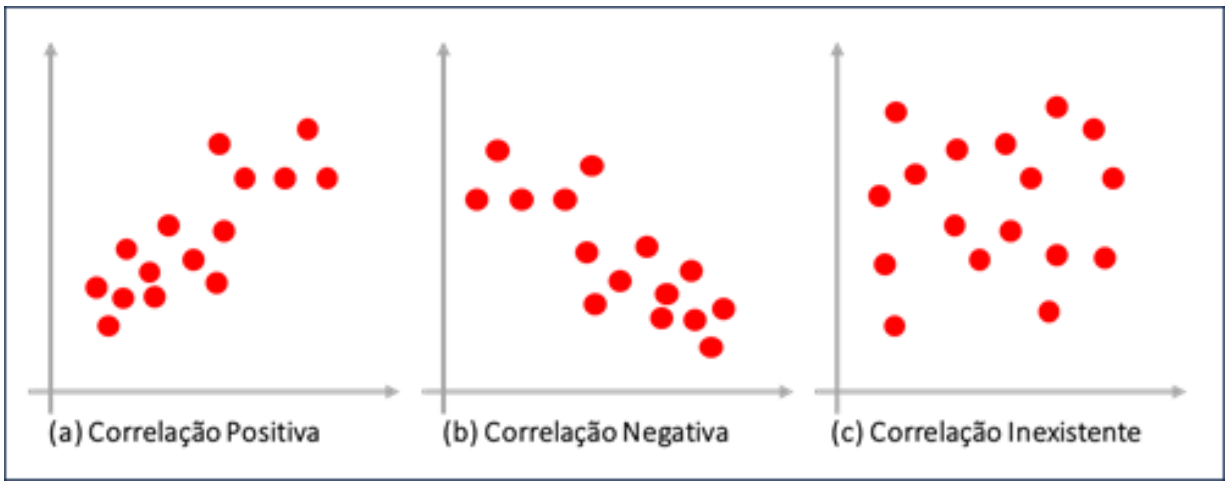


Figura 11.39: Tipos de correlação.

Correlação é a medida padronizada da relação entre duas variáveis. A correlação sempre está entre os valores 1 e -1. Uma correlação próxima a zero, conforme o gráfico (c) da figura 11.39, indica que as duas variáveis não estão relacionadas. Neste caso não é possível usar análise de regressão para a predição da variável dependente.

Uma correlação positiva, conforme o gráfico (a) da figura 11.39, indica que as duas variáveis se movem juntas, ou seja, quando o valor de uma variável aumenta, o valor da outra também aumenta. Essa relação é forte quanto mais a correlação se aproximar do valor 1.

Uma correlação negativa, conforme o gráfico (b) da figura 11.39, indica que as duas variáveis se movem em direções opostas, ou seja, quando o valor de uma variável aumenta, o valor da outra diminui. Essa relação é forte quanto mais a correlação se aproximar do valor -1.

Para concluir, pode-se afirmar que a análise de regressão é um método que utiliza a relação entre duas ou mais variáveis quantitativas de tal forma que uma variável possa ser estimada a partir de outra ou outras, mas isso não implica causalidade. Ou seja, porque existe correlação entre duas variáveis não significa que uma

é a causa da outra. Então, podemos concluir que correlação não implica causalidade.

"***Devido ao fato de (A) acontecer juntamente com (B) não significa que (A) causa (B).***"

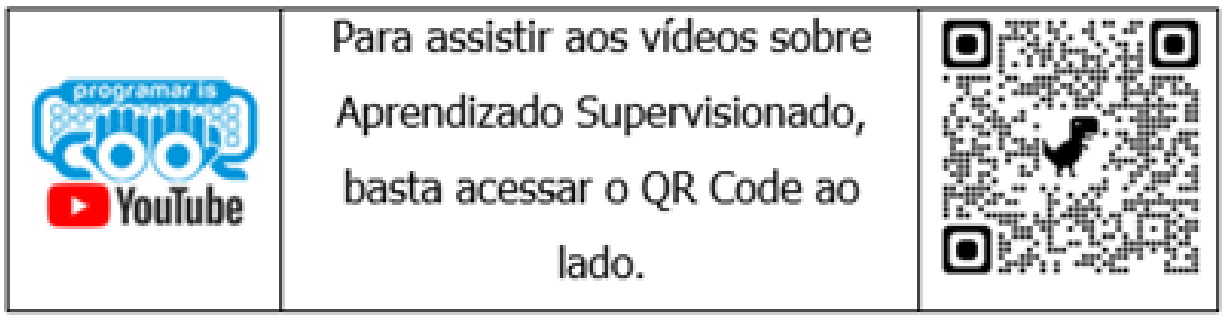


Figura 11.40: Vídeos sobre Aprendizado Supervisionado. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cIDEpjYgSjGrxKRuQUQ-_OPG

Referências

Data Science Academy - DSA. *Deep Learning Book*, 2022. Disponível em: https://www.deeplearningbook.com.br/. Acesso em: 15 abr. 2024.

Capítulo 12
Aprendizado não supervisionado

Neste capítulo serão apresentadas as duas principais técnicas de aprendizado não supervisionado: regras de associação e agrupamento. Cabe enfatizar que, da mesma forma que foi explicado no capítulo anterior, não há a necessidade de dominar as funções e fórmulas matemáticas para ser considerada uma pessoa alfabetizada em dados. Porém, tal como no capítulo anterior, sugiro que você faça pelo menos uma passagem superficial pelo conteúdo mais técnico.

12.1 Regras de Associação

Regras de associação são um método de aprendizado não supervisionado utilizado para identificar relações frequentes entre itens em conjuntos de dados transacionais. O objetivo é descobrir associações entre itens que tendem a ocorrer juntos.

Um exemplo muito comum ocorre quando um usuário da Amazon, após buscar por um livro e decidir comprá-lo, o site da empresa apresenta opções de livros similares ou complementares para serem adquiridos na mesma transação comercial utilizando a frase "*Frequentemente comprados juntos*". Isso é possível devido ao fato de a Amazon ter um volumoso conjunto de dados históricos a respeito de vendas já realizadas. O objetivo da utilização de regras de associação nessa empresa é identificar padrões de compra dos clientes para otimizar as suas vendas.

De maneira geral, a tarefa de associação tem como objetivo encontrar elementos que ocorrem simultaneamente com outros elementos em uma mesma transação, ou seja, encontrar

relacionamentos ou padrões frequentes entre conjuntos de exemplos. Portanto, Regras de Associação representam padrões existentes nas transações de um banco de dados transacional.

Uma regra assume a forma `if X then Y`:

- A parte esquerda `X` é denominada condição, premissa, antecedente, cauda ou corpo da regra.
- A parte direita `Y` é denominada conclusão ou cabeça da regra.

Nesse contexto, um *itemset* consiste em um conjunto composto por um ou mais itens que ocorrem juntos em uma transação ou evento. O número de condições determina a dimensão de um *itemset*:

- Um *itemset* contendo teste em um único atributo é denominado *1-itemset*.
- Um *itemset* contendo testes em dois atributos é denominado *2-itemset*.
- Um *itemset* contendo testes em n atributos é denominado *n-itemset*.

Uma regra de associação tem a forma `X->Y`, onde `X ={x1, x2, …., xn }` e `Y = {y1,y2, …., yn}` são conjuntos de items.

Para que uma regra de associação seja relevante, a regra deve satisfazer no mínimo a um determinado valor de suporte e confiança. O suporte representa a frequência com que um *itemset* ocorre em relação ao total de transações em um conjunto de dados. O suporte de um *itemset* qualquer é definido como o número de transações que contêm esse *itemset* dividido pelo número total de transações.

A confiança corresponde à probabilidade condicional de que um item `Y` esteja presente em uma transação, dado que o *itemset* `X` também está presente. É calculada como o suporte do *itemset* `{X, Y}` dividido pelo suporte do *itemset* `X`.

Outro parâmetro importante é o *lift*, que mede a importância de uma regra de associação, indicando a probabilidade de os itens em uma regra de associação ocorrerem juntos, em comparação com sua ocorrência esperada se os itens fossem independentes uns dos outros. Um *lift* maior que 1 indica uma associação mais relevante do que seria esperado ao acaso.

As figuras 12.1, 12.2 e 12.3 apresentam um conjunto de dados e uma regra de associação proposta com a respectiva confiança, suporte e lift.

Figura 12.1: Suporte de uma Regra de Associação.

A figura 12.1 apresenta um conjunto de dados com cinco transações realizadas em um banco de dados relacional relativo a vendas de produtos alimentícios. Cada linha descreve uma venda composta pelos atributos `Transação`, `Leite`, `Pão`, `Manteiga` e `Café` que representam, respectivamente, o identificador da transação e os produtos que foram vendidos nessa transação. Na primeira linha do conjunto de dados, o número 1, na coluna `Transação`, representa o identificador da transação. Nas demais colunas, o número 1 indica que o produto foi vendido na transação, por outro lado, o número 0 indica que o produto não foi vendido na transação. Assim, na transação 1 foram vendidos os produtos leite e pão.

A regra de associação proposta nessa figura é `{leite, pão} -> {manteiga}` (se leite e pão, então manteiga) que significa o seguinte: "se leite e pão são vendidos, então manteiga é vendida junto". Assim, o suporte dessa regra, cujo *3-itemset* é `{leite, pão, manteiga}`, pode ser calculado como o número de transações que

contêm os três itens do *itemset* dividido pelo número total de transações. Devido ao fato das 5 transações existentes, só a transação 4 possuir os três itens, o suporte será igual a 1/5 = 0,2, ou seja, 20%, conforme apresentado na figura 12.1.

A confiança dessa regra é calculada por meio da divisão do valor do suporte da regra completa, apresentado anteriormente (0,2), pelo suporte do *2-itemset* `{leite, pão}`. Como esse *itemset* ocorre nas transações 1 e 4 num total de 5 transações, conforme apresentado na figura 12.2, o suporte é igual a 2/5 = 0,4, o resultado da confiança da regra será 0,2/0,4 = 0,5.

Transação	Leite	Pão	Manteiga	Café
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

Regra: {leite, pão} → {manteiga}

- Confiança = 0,2/0,4 = 0,5 (50%)

Figura 12.2: Confiança de uma Regra de Associação.

O *lift* dessa regra é calculado dividindo a confiança da regra completa (0,5) pelo suporte do *1-itemset* `{manteiga}`. Devido ao fato de duas transações possuírem o item `manteiga`, conforma apresentado na figura 12.3, o suporte desse *itemset* será 2/5 = 0,4. Assim, o *lift* da regra será = 0,5/0,4 = 1,25.

Transação	Leite	Pão	Manteiga	Café
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

Regra: {leite, pão} → {manteiga}

- $\text{Lift} = \frac{\text{Confiança \{leite, pão\} => \{manteiga\}}}{\text{Suporte \{manteiga\}}}$
- Lift = 0,5/0,4 = 1,25

Figura 12.3: Lift de uma Regra de Associação.

A partir do exemplo apresentado anteriormente, é possível imaginar a complexidade de se encontrar regras de associação relevantes em ambientes com milhões ou trilhões de transações com vários atributos como é o caso da Amazon, Netflix e o facebook. Nesses casos, o suporte, a confiança e o *lift* devem ser utilizados como filtros importantes para selecionar regras de associação relevantes.

Na Amazon, as regras de associação podem ser utilizadas para sugerir produtos a compradores a partir de regras do tipo `quem compra x, compra y`, ou na Netflix para sugerir filmes a usuários a partir de regras do tipo `quem vê o filme z, vê o filme w`, ou no Facebook para sugerir amizades entre usuários a partir de regras do tipo `quem é amigo de p, é amigo de q`.

12.2 Agrupamento

A análise de agrupamento, também denominada **clusterização** ou *clustering*, é uma técnica de aprendizagem não supervisionada, o que significa que não requer rotulação preliminar dos dados, diferentemente da classificação que é uma técnica supervisionada na qual as classes dos dados são estabelecidas previamente. Assim, o agrupamento é usado para a descoberta de conhecimento, ao invés de fazer predições.

A clusterização é amplamente utilizada em várias aplicações de Ciência de Dados, como:

- Na medicina, para a identificação de categorias de diagnósticos, pacientes e remédios.
- Na biologia, para categorizar animais e plantas.
- Na agricultura, para categorizar verduras, grãos, frutos e solos.
- Em marketing, para identificar grupos de clientes, produtos e serviços.
- Em finanças, para identificar tipos de fraudes.

Essa técnica é utilizada para explorar e compreender a estrutura interna de conjuntos de dados, identificar grupos (clusters) de interesse e oferecer subsídios para a tomada de decisões com base nas características dos grupos identificados.

Os algoritmos de clusterização examinam as características dos dados e agrupam os itens com base na proximidade ou similaridade entre eles, utilizando medidas de distância ou similaridade, como a distância Euclidiana, distância de Manhattan, distância Hamming, entre outras. Os métodos de agrupamento visam agrupar objetos similares entre si e diferentes de objetos pertencentes a outros grupos.

Existem vários algoritmos de clusterização, dentre eles os hierárquicos e os particionais, cada um com suas próprias abordagens e critérios de formação de grupos. Entre os algoritmos de clusterização mais comuns está o *K-means* (K-médias), que é um algoritmo particional que recebe um número de partições proposto pelo cientista de dados, o K, e um conjunto de dados como entrada e procura encontrar K grupos disjuntos de dados.

De maneira simplificada, os passos do algoritmo *K-means* são:

- Escolha a quantidade de clusters aleatoriamente, ou seja, o valor de K;
- Escolha aleatoriamente um ponto ou determine um item do conjunto de dados para representar o centroide (ponto em que as coordenadas são as médias das coordenadas dos elementos de um grupo de dados) de cada um dos K clusters;
- Repita:
    - Calcule a distância de cada elemento do conjunto de dados aos centroides;
    - Atribua cada ponto ao centroide (ponto médio de todos os elementos que pertencem a um mesmo cluster) mais próximo formando um cluster;

    - Recalcule o centroide para cada cluster;
- Até ter estabilidade.

A figura 12.4 apresenta o diagrama de dispersão de um conjunto de dados para um exemplo de aplicação da análise de agrupamento utilizando o algoritmo *K-means*.

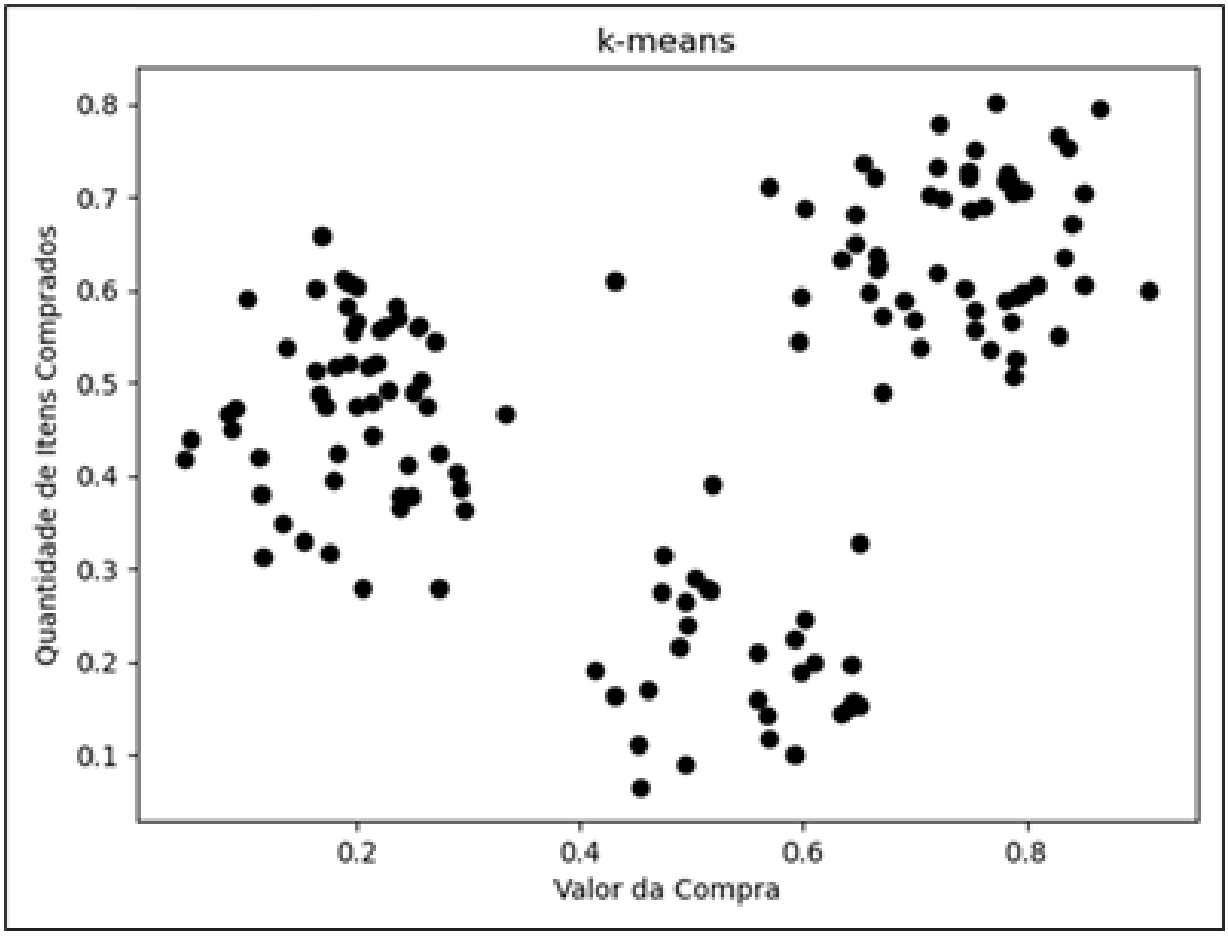


Figura 12.4: Conjunto de dados para análise de agrupamento.

Os pontos apresentados na figura 12.4 representam as vendas de um supermercado. O eixo horizontal representa o valor de uma compra de um cliente e o eixo vertical representa a quantidade de itens que compõem essa compra. Cabe ressaltar que os dados foram padronizados para ficarem na mesma escala e ocuparem valores entre 0 e 1.

O primeiro passo para aplicarmos o algoritmo *K-means* consiste na escolha da quantidade de clusters, ou seja, o valor de K. Considerando o diagrama da figura 12.4, pode-se observar a tendência de existirem 3 clusters, por isso escolheremos o $K = 3$. Cabe ressaltar que em projetos reais de ciência de dados utiliza-se recursos matemáticos e estatísticos para auxiliar a escolha do valor de K, além disso, usa-se vários valores de k para verificar o melhor resultado.

O segundo passo compreende em escolher os três centroides, pois $K = 3$. Na primaira iteração, serão escolhidos três pontos aleatoriamente representados pelos retângulos da figura 12.5. Nas próximas iterações, os centroides serão calculados por meio da média aritmética das coordenadas dos elementos que pertencem ao mesmo grupo.

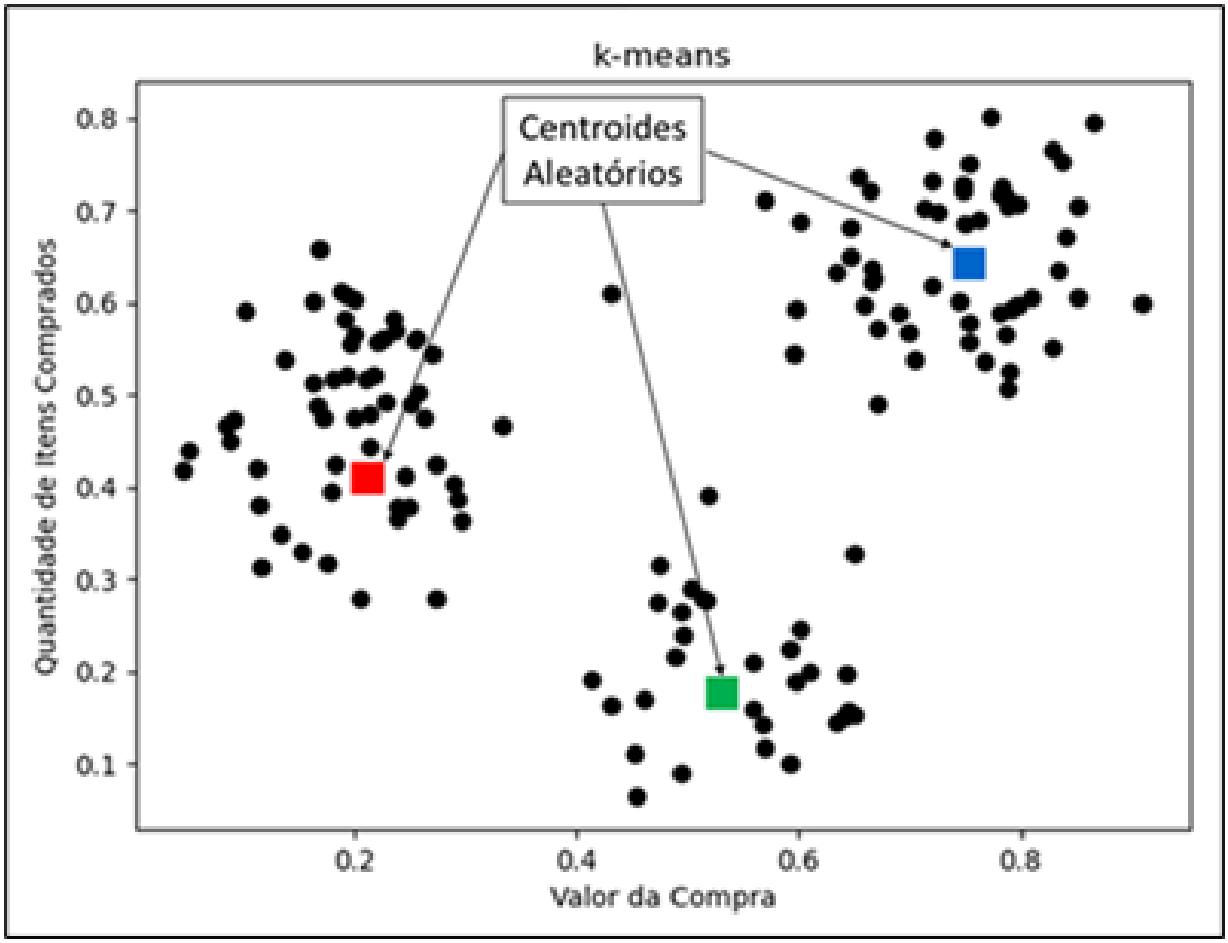


Figura 12.5: Conjunto de dados com centroides.

Os próximos passos serão executados iterativamente até atingir a estabilidade. A primeira etapa da iteração consiste em calcular a distância de cada elemento do conjunto de dados aos centroides, optou-se por utilizar a distância euclidiana para o presente exemplo, conforme apresentada na figura 12.6. Nesta figura, o círculo representa um item do conjunto de dados e os retângulos representam os centroides escolhidos, para descobrir a qual grupo o item de dado pertencerá, calcula-se a distância euclidiana do ponto a cada centroide, essas distâncias estão representadas pelas linhas cheias na figura.

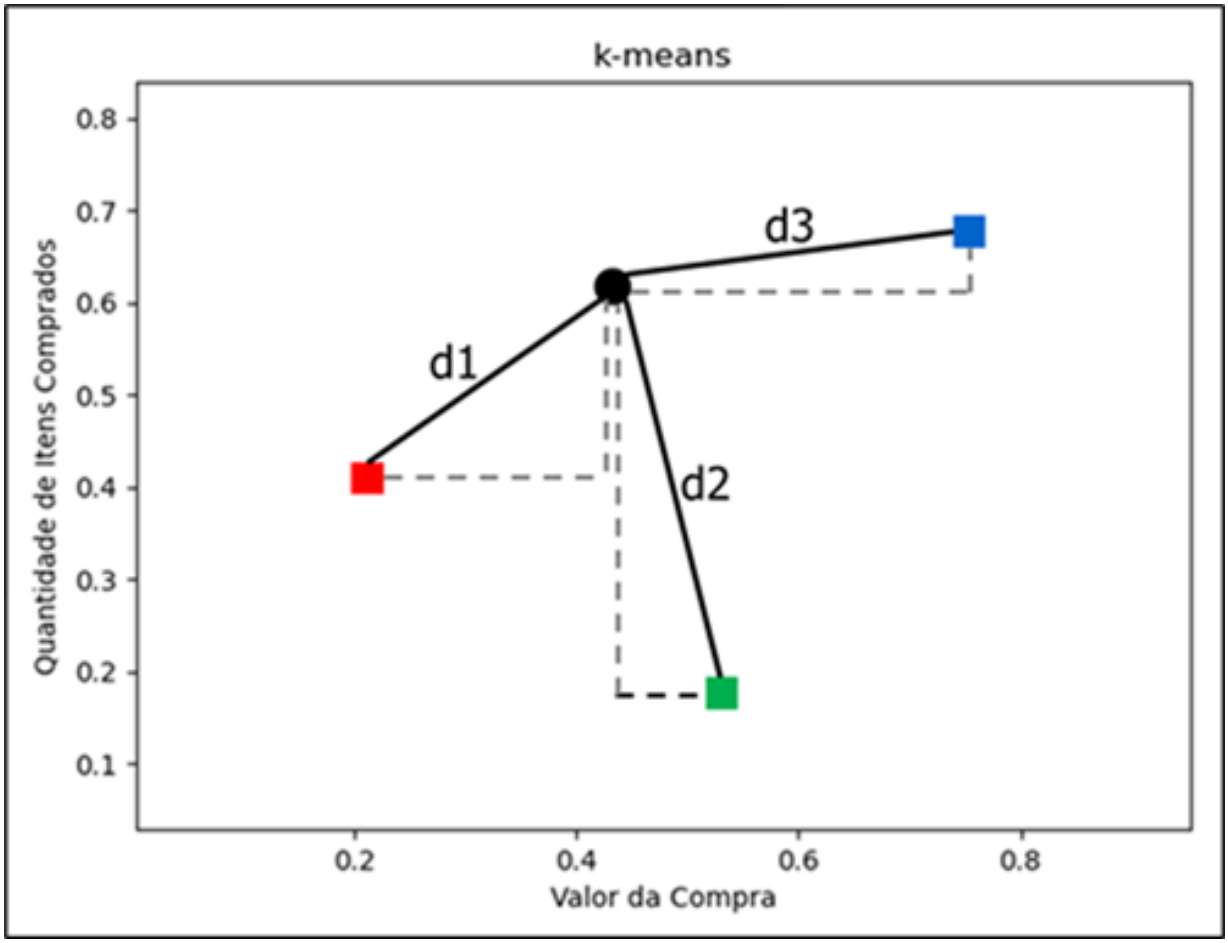


Figura 12.6: Distância euclidiana de um ponto aos centroides.

Como pode ser observado na figura 12.6, o item de dado escolhido está mais próximo ao centroide mais à esquerda. Então, a segunda atividade da iteração consiste em atribuir esse item e todos os outros que estiverem mais próximos a esse centroide a um grupo, será feito o mesmo procedimento para todos os itens de dados, formando inicialmente três grupos fundamentados nos centroides.

Após a formação desses grupos, o terceiro passo da iteração consiste em recalcular as coordenadas do centroide para cada grupo formado. Neste caso, serão duas coordenas: o eixo horizontal que será a média aritmética do valor da compra dos elementos dentro do cluster e o eixo vertical que será a média aritmética da quantidade de itens comprados dos elementos dentro do mesmo cluster. Caso

esse novo centroide calculado coincida com o centroide anterior, o algoritmo finaliza a execução.

Caso o conjunto de dados seja muito grande, exigindo muitas interações, o cientista de dados poderá estabelecer um número máximo de iterações para o algoritmo finalizar a sua execução tornando a técnica viável.

A figura 12.7 apresenta os grupos resultantes da análise de agrupamento.

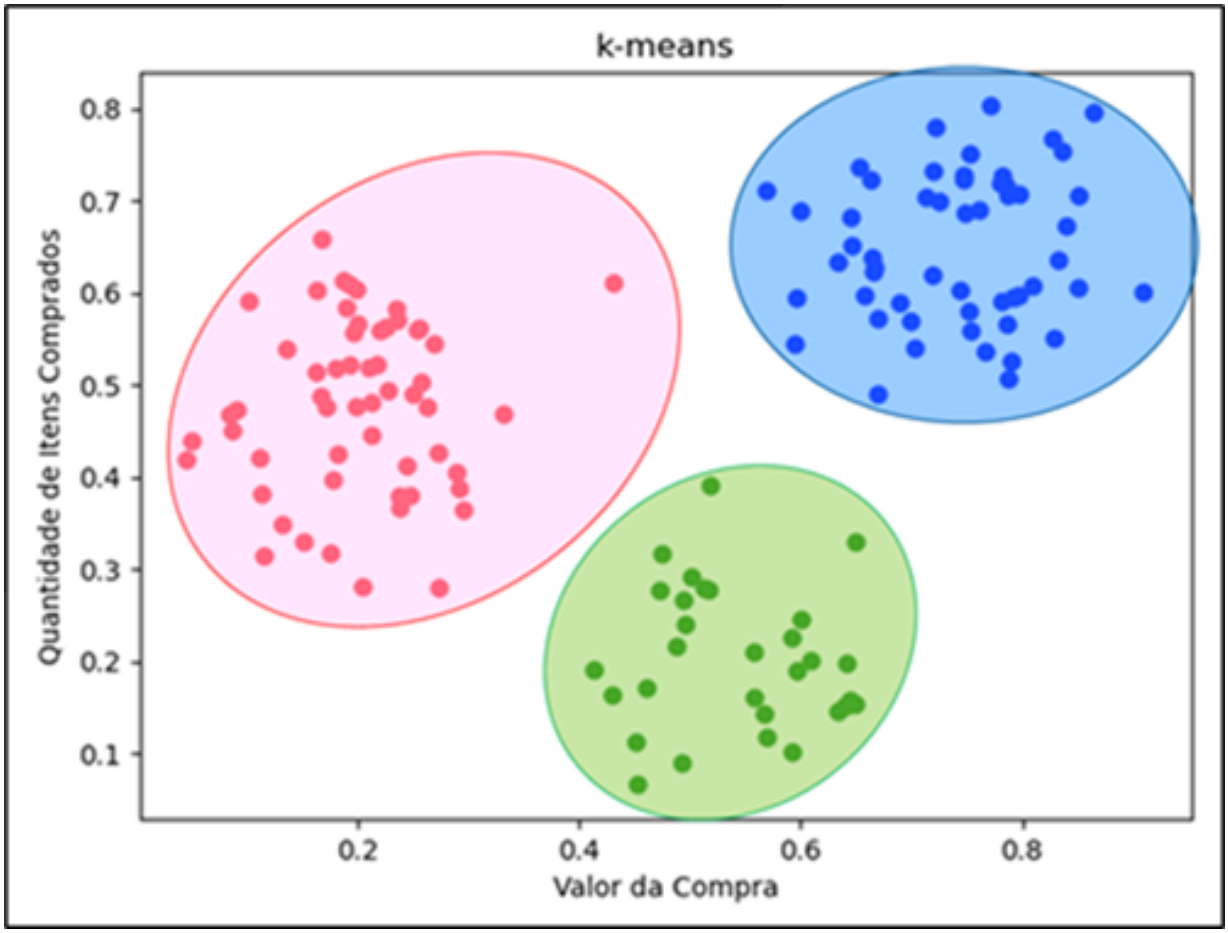


Figura 12.7: Grupos resultantes da análise de agrupamento.

Como pode ser observado na figura 12.7, a aplicação da técnica de agrupamento ao conjunto de dados do exemplo resultou em três grupos de clientes do supermercado. O próximo passo consiste em

apresentar ao dono do negócio esses grupos para que novas perguntas sejam feitas e respondidas, por exemplo:

- Qual o valor mínimo, médio e máximo gasto por compra em cada grupo?
- Qual tipo de produto cada grupo costuma comprar (verdura, carne, peixe etc.)?
- Quais são os cinco produtos mais comprados por grupo?
- Qual ou quais os dias da semana cada grupo costuma ir ao supermercado?

Assim, torna-se possível gerar novos conhecimentos sobre o negócio a fim de subsidiar decisões que venham a otimizar as vendas do supermercado analisado.

Cabe ressaltar que no exemplo analisado foram utilizadas apenas duas dimensões para cada item de dados, porém em projetos reais de agrupamento são utilizadas dezenas ou mais dimensões.

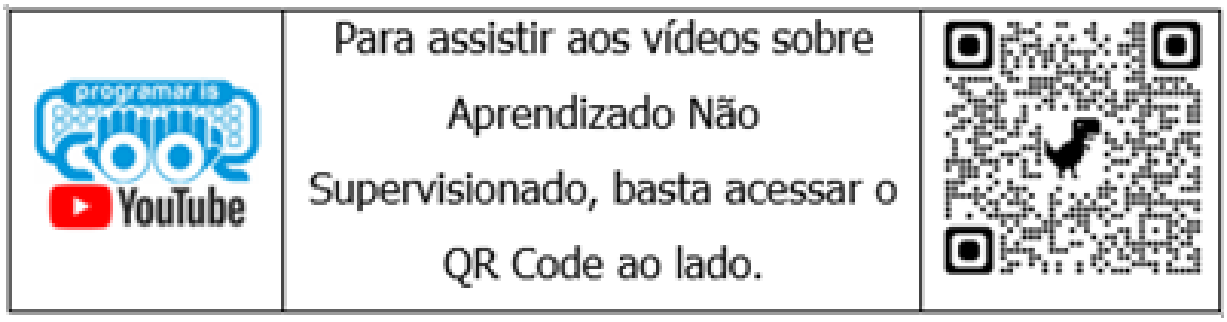


Figura 12.8: Vídeos sobre Aprendizado Não Supervisionado. Link: https://www.youtube.com/playlist?list=PLdoTFRH60cICzd4kYYuHBRWGff2wBAzzz

CAPÍTULO 13
Considerações finais

Existem as organizações cujas lideranças tomam suas decisões baseadas em dados e outras cujas lideranças tomam suas decisões baseadas no "achismo" ("eu acho que...").

Este livro foi concebido focando nesses dois tipos de gestores, com o objetivo de aprimorar a estratégia decisória do primeiro e transformar a do segundo, por meio da compreensão dos dados organizacionais, pois acredito que todos nós podemos e devemos evoluir para nos adaptar a novos contextos, principalmente quando nos voluntariamos ou somos convidados a coordenar equipes, seções, departamentos ou organizações dos mais diversos tipos, tamanhos ou áreas de atuação em um ambiente em constante mudança, como é o ambiente corporativo atual.

Para atingir esse objetivo, torna-se necessário o entendimento da linguagem dos dados, de como os dados devem ser organizados e como podem ser transformados para serem usados no apoio à decisão nas organizações. Ou seja, torna-se necessário o processo de alfabetização de dados.

Aqui, apresentei os assuntos mais relevantes relacionados a esta jornada. Vimos inicialmente o conceito de organizações orientadas a dados e a importância dos dados para os negócios. Em seguida, passamos para o conceito de alfabetização de dados, a sua importância para as organizações orientadas a dados e as habilidades mais importantes a serem desenvolvidas por profissionais para serem considerados alfabetizados em dados.

Para o desenvolvimento dessas habilidades, vimos as definições de sistemas transacionais, sistemas de apoio à decisão e respectivos usos. Abordamos projetos de banco de dados, bancos de dados relacionais, normalização de modelos relacionais, fundamentos da

linguagem SQL, Big Data e os bancos de dados NoSQL. Mostramos, também, o conceito de Data Warehouse, a modelagem dimensional e exemplos de consultas OLAP.

Com o objetivo de desenvolver a habilidade de ler e o entender códigos, abordamos os fundamentos de lógica de programação e como esses fundamentos são implementados na linguagem de programação Python. Por último, apresentamos os conceitos de Ciência da Dados e suas técnicas, dando-lhe mais ferramentas para assimilar os fundamentos sobre os modelos descritivos e preditivos para o seu uso no apoio à tomada de decisões.

Acredito que com esses conhecimentos, você poderá ser considerado um(a) profissional alfabetizado(a) em dados capaz de tomar decisões embasadas em dados.

Finalmente, para ampliar o objetivo deste livro, gostaria de fazer uma última citação, atribuída a de Abraham Lincoln: "*A maior habilidade de um líder é desenvolver habilidades extraordinárias em pessoas comuns*".

Corroborando com Lincoln, é possível observar que nenhum líder nasce pronto e que os líderes de hoje já foram pessoas comuns que ao longo do tempo adquiriram habilidades extraordinárias, normalmente sob a orientação de alguma pessoa que já possuía essas habilidades, um líder.

Se você, caro leitor ou leitora, ainda não é líder, este livro ajudará você a desenvolver algumas das habilidades extraordinárias de um líder, principalmente a tomar decisões embasadas em dados.

Se você já é líder e se desafiou a desenvolver ou aprimorar as habilidades de compreender, organizar e interpretar os dados da sua corporação por meio da alfabetização de dados, gostaria de parabenizá-lo por ainda possuir o hábito de incrementar suas habilidades extraordinárias.

No entanto, ressalto que como líder, segundo Lincoln, você deve ter a capacidade e o compromisso de desenvolver habilidades extraordinárias em pessoas comuns, ajudando a formar novos líderes, mantendo o ciclo virtuoso da liderança, ou seja, de líderes que formam líderes.

Então, sugiro a vocês, líderes e futuros líderes que leram este livro, que assistiram aos vídeos do canal *@ProgramarIsCool*, que se divertiram com a nossa didática e concordaram com a importância da alfabetização de dados para a gestão das organizações, que iniciem a desenvolver habilidades relacionadas ao tratamento dos dados em seus colaboradores por meio deste livro. Assim, elas desenvolverão habilidades extraordinárias e, quem sabe, um dia se tornarão líderes também.